UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

Federated Learning for Indoor Human Activity Recognition: Adapting to Changing Realistic Environments

Martijn M. van der Linden

M.Sc. Thesis
May 2023

Examiners:
prof. dr. P. Havinga
J. Klein Brinke, MSc
dr. Y. Miao

Pervasive Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Acknowledgements

This thesis has been conducted at the Pervasive Systems group under guidance of my daily supervisor Jeroen Klein Brinke. I want to thank him for his supervision in the past year, which helped me explore my own interests and the direction of my research. I am grateful for the many talks I have had with him during meetings, coffee breaks and everything in between to solve my issues or brainstorm about areas of interest. I would also like to thank the graduation committee consisting of prof. dr. Paul Havinga and dr. Yang Miao, who have provided me useful feedback to finalise my work.

To all participants of my experiments: thank you for your cooperation and flexibility during the data measurements, even though things did not always go as smoothly as planned. My thesis would not have been complete without your help.

Thank you to my parents for supporting me throughout my thesis and giving me suggestions to continue or improve my work. A special thanks goes to my close friends, whose discussions, constructive feedback, and support was absolutely sublime.

Abstract

This work focuses on the optimisation of federated learning (FL) for indoor human activity recognition (HAR) in dynamic environments using Wi-Fi-based channel state information (CSI) signals and resource-constrained devices. This is achieved by analysing the impact of various parameters to deal with unseen activity locations and dynamic client participation. Promising advancements are found in the amount and type of communication and local computation of clients, as well as the use of existing models to increase convergence speed of models based on unseen data. Traditional HAR methods can be intrusive, are limited by environmental constraints and invade ones privacy. FL, a distributed learning technique, allows for collaboration between multiple clients with each their own unique data, which maintains privacy while allowing for a powerful machine learning (ML) model. The main limitation of FL for HAR found in previous work is the lack of realistic testing environments. By simulating scenarios with resource-constrained devices, this study explores the impact of local computation, varying aggregation algorithms, limited data availability and changing environments. The research highlights opportunities and challenges of realistic environments based on extensive analysis and comparison of the varying parameters, providing recommendations to maximise the benefits from using FL to train a deep learning model for indoor HAR. The thesis concludes with an outlook for the future direction of this system and an overview of the remaining challenges to be addressed.

Contents

Acronyms

ADL	activities of daily living
CSI	channel state information
DNN	dense neural network
FL	federated learning
FMCW	Frequency Modulated Carrier Wave
HAR	human activity recognition
IID	identically and independently distributed
IoT	Internet of Things
MIMO	multiple inputs multiple outputs
ML	machine learning
MSE	mean squared error
OFDM	Orthogonal Frequency Division Multiplexing
RSSI	Received Signal Strength Indicator
SGD	stochastic gradient descent
TFF	TensorFlow Federated
USRP	Universal Software Radio Peripheral

List of Figures

List of Tables

Chapter 1

Introduction

The ability to automatically monitor activities carried out by people is of interest for applications such as smart homes, security surveillance, and elderly care [3]. With such human activity recognition (HAR), activities of daily living (ADL) can be monitored and automatically classified without the need for wearable sensors or video material, allowing for non-intrusive and private activity recognition, respectively. The focus of this thesis is on elderly care specifically, but the applied methods can be extended to other HAR applications. In elderly care, it is desirable to be able to monitor the activity of a person at all times with confidence. A dangerous situation should be recognised immediately to ensure the safety of the elderly. Various HAR techniques have been developed, including methods based on computer vision, wearable motion sensors, and acoustic-based methods [3, 4]. These methods are either intrusive (e.g. inconvenience of a wearable sensor), are limited by their environmental constraints (e.g. lack of proper lighting conditions), or invade a person's privacy (e.g. cameras inside a home). Furthermore, these techniques require additional hardware, and thus an adaption of the existing environment. A wireless method operating in a non-intrusive manner is desired.

Performing wireless HAR in indoor environments is conveniently done using Wi-Fi signals [3], which allows for non-intrusive monitoring built on low-cost communication channels [5]. The measured signals are channel state information (CSI) data, which are complex and contain a large amount of data, especially when the number of performed actions increases [5]. Machine learning (ML) can provide a solution by training a deep learning model which is capable of recognising and classifying activities. The problem then is collecting suitable data, as a model trained on data from a single location does not generalise well for new, never observed environments. A naive, time-consuming approach would be to gather data separately in different locations, which is especially cumbersome when new locations are introduced and a new model needs to be trained to fit the new data. When devices at different locations have access to differently distributed data corresponding to their unique physical environment, a better solution would be to use federated learning (FL). In this distributed learning technique, an ML model is trained through the collaboration of multiple clients[1], all of which train a model based only on locally available data. Each client calculates the update to the model and communicates only this update to either a central server (centralised FL) or to peers in the network (decentralised FL), depending on the specific setup. All individual updates are aggregated and form the update to the general model, which is then communicated back to the clients. This way, clients keep their own distinctive data from discrete locations and only the model itself is to be shared, while benefiting from features of data from all nodes and allowing models to be trained in parallel. Data containing information on a person's activity can be kept at clients, which maintains privacy and reduces communication. This results in a potentially faster converging and more powerful ML model compared to conventional ML [5]. Training a deep learning model for HAR is especially interesting for FL due to the fact that HAR is a relatively simple problem and therefore requires smaller datasets than, for example, image recognition applications [6]. These smaller datasets can be easily handled by small or Internet of Things (IoT) devices that can also be used as nodes in FL. In an indoor setting, these devices are easily integrated into the environment, e.g. as smart devices, Wi-Fi access points, or smartphones, making FL suitable for training an HAR model for indoor classification. An additional advantage is the variation in the data measured between devices, which adds to the ability of the machine learning model to generalise.

Other developments in FL include the introduction of personalised models [7, 8], and the efficient grouping of participating devices [7, 9]. With these developments, new applications have been made possible and FL frameworks are designed specifically for HAR. An example where FL currently finds its real-life use is in mobile keyboard development from e.g. Google [10], where data from users' keyboard interaction are used in a federated setting to train a general model and improve prediction on user input. When selecting participating clients, only phones that meet the participation requirements are selected: phones must be idle, charged, and connected to Wi-Fi. This implementation shows a key feature of FL: maintaining privacy while learning from locally generated data from various different environments.

1.1 Problem Statement

The use of CSI-based data for HAR in FL is not a new concept. Previous studies have demonstrated the feasibility of using CSI data to classify indoor activities and the added benefits of FL in this context. However, several limitations still exist, as the conditions in previous work are not representative of real-life scenarios: the relative location of the performed activity to the measuring devices is controlled, and the communication channels on which the activities are measured are fixed and predetermined.

Although efforts have been made to simulate realistic transmitter and receiver locations, research on analysing the effect of a change in device participation and activity location specifically aimed at HAR is lacking. Furthermore, extensive research has been conducted to determine the effects of various parameters in the FL frameworks, such as the type

[1]The terms clients, nodes and devices mean the same in this context and are used interchangeable throughout this work.

of averaging algorithm, the trade-off between communication and computation, and the design of the FL network together with its communication methods. However, determining the direct influence of these parameters in a scenario where CSI-based HAR operates in a real-life indoor environment remains an open challenge.

To address these limitations, this thesis aims to investigate the challenges associated with CSI-based HAR in realistic indoor environments using FL. Specifically, the study examines the effects of varying transmitter and receiver locations on the performance of the HAR system, a varying location of human activity, a change in participating clients, limited data availability, and other realistic scenarios as tested in a simulated indoor environment using IoT devices with resource constraints. The goal of this study is to identify and empirically test the influence of these parameters to create a robust, flexible, and efficient system that can automatically classify human activities in a constantly changing environment, with uncertainties about human behaviour and limitations on communication and computation in participating devices.

To reach this goal, the following research question is stated:

> *How is CSI-based human activity recognition improved with the use of federated learning approaches, aimed towards practical use in changing environments with changing activity locations and client participation?*

This research question is answered by means of the following sub-questions, of which hypotheses are given:

1. **How does distributed computing using federated learning improve indoor human activity recognition on IoT devices compared to centralised machine learning approaches?**
   It is hypothesised that FL can increase HAR performance while allowing for a decrease in device resources compared to non-distributed learning.

2. **Which federated learning settings can be used to maximise the performance of human activity recognition?**
   Finding the optimal combination of FL aggregation algorithms, local computation strategies, and model hyperparameters maximises the classification potential for HAR.

3. **How can a change in activity location be dealt with to create a location-independent federated learning system for human activity recognition in dynamic environments?**
   By incorporating model adaptation based on unseen data, the classification model for human activity recognition is readjusted within reasonable time and limited resource consumption to adequately classify activities in the new location.

4. **How can dynamic client participation be dealt with in federated learning for human activity recognition to ensure sufficient classification performance in dynamic environments?**
   Model adaptation can be reliably implemented to adapt an already existing classification model to unseen data from new clients within reasonable time and limited resource consumption.

5. **What trade-offs exist between model complexity, classification performance, and resource constraints in human activity recognition for IoT devices and how can these trade-offs be optimised?**
   Increasing the model complexity improves the classification performance but imposes a greater amount of required device resources. To optimise performance within these limitations, leveraging FL with suitable aggregation algorithms and fine-tuning the amount of local computation can yield optimal results.

Sub-question 1 is answered through an extensive literature study and an empirical comparison between the performance of federated and centralised machine learning models. Question 2 is answered by optimising and comparing different classification models through varying parameters, using data labeled as the *complete* dataset. Questions 3 and 4 are answered by simulating environmental changes to withhold and later introduce certain data from network optimisation. This is done with the use of the *location* and *client* dataset, respectively, allowing for convenient introduction of unseen data. Question 5 is answered by varying the required amount of resources and comparing different performance outcomes of the classification models. The combined results of all datasets is used to anwer this question.

The main contributions of this thesis are:

- A comparison between FL and centralised ML for CSI-based indoor HAR
- A comparison between different FL aggregation algorithms for CSI-based indoor HAR
- A comparison between various amount of local computation by clients participating in an FL framework
- The effect of adding new activity locations to an existing FL network
- The effect of adding new participating clients to an existing FL network

In all of these contributions, resource-constrained devices are taken into account.

The research questions are answered throughout this work. The remainder of this thesis is structured as follows: Chapter 2 describes the relevant theory regarding ML, FL, and HAR. Subsequently, Chapter 3 investigates the current state of the art of FL for human activity recognition. Chapter 4 presents the approach and results of a pilot study aimed at gaining preliminary insights. Chapter 5 describes the data collection and preprocessing methods for the data used in the rest of the work. Then, Chapter 6 describes in detail the methodology applied during this research. Subsequently, Chapter 7 presents the results of the experiments carried out as described in the methodology, which are then discussed in Chapter 8. Finally, the work is concluded in Chapter 9 and recommendations for future work are given.

Chapter 2

Background

This chapter provides an overview of the key concepts and theories relevant to the research. It begins by describing the fundamentals of neural networks, followed by a description of FL. Then, the concepts behind wireless human activity recognition are explored. Furthermore, this chapter investigates the role of device constraints in the context of distributed machine learning. Understanding these concepts is of importance to understanding the following chapters and the contributions made in this thesis.

2.1 Neural Networks

This section highlights the theoretical foundation of neural networks. The fundamental mechanics of neural networks are explained, as well as common factors that can cause a reduction in model accuracy.

2.1.1 Concepts of neural networks

Neural networks are a type of machine learning models that are inspired by the neural structure of the human brain. In neural networks the nodes, or neurons, are interconnected and cooperate to learn relationships between input and output data. Between neurons, the data goes through a function that modifies the data in such a manner that the final output is closer to the expected output. This *activation function* $\Phi(\cdot)$ is denoted as follows for output $\hat{y}$ [11]:

$$\hat{y} = \Phi(\mathbf{w} \cdot \mathbf{x}) \tag{2.1}$$

with weight vector $\mathbf{w}$ and input data $\mathbf{x}$. A neural network can consist of many of such connections. The weights in $\mathbf{w}$ are variables influencing the predicted output. Therefore, training or optimising a neural network is a process of finding the set of weights that result in the desired output for a given input data. A schematic overview of a neural network model is given in Figure 2.1.

The type of activation function influences the network's output. Examples are the *sigmoid* function, which maps outputs to a probability of observing that output, or the

ReLU function, which maps weights lower than 0 to 0, and weights higher than 0 are unchanged. Combining activation functions across different layers in the neural network allows for complex connections between input and output [12, 13].

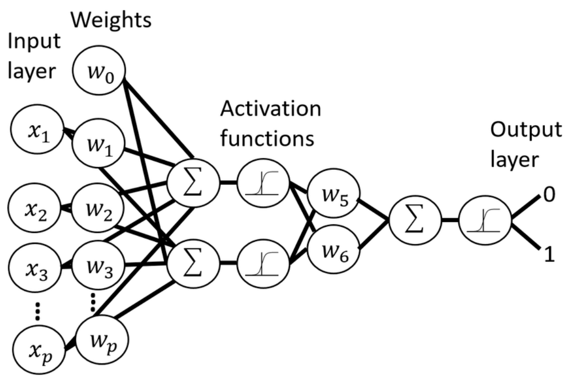


Figure 2.1: Schematic overview of a neural networks' architecture, each set of weights and activation functions forming a layer [1].

To optimise the weights $\mathbf{w}$ in a network, a weight vector is to be found that minimises the loss function $E(\mathbf{w})$, which evaluates a network's ability to correctly classify the data by determining the error between the predicted and actual output. After optimisation, a proper neural network model is then able to correctly classify or make predictions based on the input data during inference, i.e. applying the trained model to a dataset. The specific choice of loss function depends on the application and the type of data being considered. The weights are updated through iterative optimisation steps in which the gradient of the loss function with respect to the weights is computed. This is called *backpropagation.* Then, in each step, based on the loss gradient the weights are adjusted in the direction which reduces the loss and thus results in a more accurately performing neural network. This step is called *gradient descent* and can be executed in many different ways. The weight update is given by:

$$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w_{old}} \tag{2.2}$$

with loss function L and learning rate α. The above is continued until the whole dataset has been iterated through. This is considered to be a single *epoch* of training. A neural network model is trained with the same dataset multiple times if the number of epochs is more than one. The pseudocode for training a neural network is given in Algorithm 1, with dataset D, epochs E, and batch size B.

One of the methods to perform gradient descent with is *stochastic gradient descent (SGD)*, in which the weight vector is updated after each data point [12]. When part of the data is considered before updating the weights, the term *mini-batch* SGD is used. An update is then computed over a subset of the data called a *batch.* As opposed to SGD, far less updates are necessary and thus less resources and computation time is needed. On the other hand, a big batch size would lead to high memory requirements due to the fact that computations are performed over a large portion of the data. Different applications thus allow for different batch sizes [11]. Besides limited memory availability, the batch size is also influenced by the desired model performance. Keskar et al. [14] note that a larger batch size results in a lower model performance. A trade-off thus exists between memory availability, computational resources, and model performance.

Algorithm 1 Neural network training.

```
1: procedure TrainNetwork($D, E, B$)
2:     Initialise network weights
3:     for $e \leftarrow 1$ to $E$ do
4:         Shuffle training dataset $D$
5:         for $b \leftarrow 1$ to $\frac{\text{length}(D)}{B}$ do
6:             Select batch $D_b$ from $D$ of size $B$
7:             Calculate network output on $D_b$
8:             Compute loss and gradients
9:             Update model weights with backpropagation
10:        end for
11:    end for
12:    return Trained model
13: end procedure
```

2.1.2 Overfitting

In machine learning, overfitting means that a model is fitted on a training dataset, while the model does not perform well on classifying a separate test dataset. The model thus does not generalise to unseen data. This can happen due to noise found in the training data which the model adapts to. Additionally, while a high number of epochs can lead to a better performing model, an excessive number of epochs could result in the model overfitting to the training data. Increasing the amount of training data might present the model with data in which the noise is absent or different, thus allowing for differently structured data. Also, decreasing a model's size can increase its ability to generalise. Lastly, the k-fold cross-validation method can be used to help against overfitting. Training data are divided into k equal parts, of which $k-1$ parts are used for model optimisation, and the last k part is used for validation. After each part has served as validation set once, the model performance is averaged based on the validation performance of the cross-validated models. This approach allows for different combinations of training data and for a more accurate estimation of the model's performance, enabling easier detection of overfitting [11, 15].

2.1.3 Regularisation

One of the techniques that can be used to help prevent a model from overfitting is the use of regularisation, which decreases the complexity of a model. A regulariser is a term that is added to the loss function and acts as a penalty. Goodfellow et al. [16] describe regularisation as "any modification we make to a learning algorithm that is intended to reduce its generalisation error but not its training error", thus counteracting overfitting.

L_1 and L_2 regularisation

Common regularisation techniques are L_1 and L_2 regularisation, of which the latter is more commonly used in machine learning. Both help reduce a model's complexity by the addition of a penalty to the loss function.

In L_2 regularisation, also known as the squared norm penalty, weight values are forced towards zero with the following regularisation term:

$$\Omega(w) = \lambda||w||_2^2 = \lambda \sum_{i=0}^{d} w_i^2 \tag{2.3}$$

with d inputs, weight w, and regularisation parameter λ. Changing λ influences the impact of the penalty on the loss function. During the gradient descent step in the training phase of the model, a parameter update is then represented by:

$$w_{new} = (1 - \alpha\lambda)w_{old} - \alpha\frac{\partial L}{\partial w_{old}} \tag{2.4}$$

with loss function L and learning rate α. The factor $(1 - \alpha\lambda)$ reduces the old weight by $\alpha\lambda$, and therefore L_2 regularisation is also commonly referred to *as weight decay*. Considering the weight update, it can be intuitively reasoned how the regularisation acts under different absolute values of the weight: a greater value of the old weight decreases the value of the new weight more rapidly, while a smaller value has less impact on the new weight value. The effect is that weight values are kept small. Smaller weight values are influenced less by the regularisation factor and thus weights are unlikely to become exactly 0. This form of regularisation acts as forgetting mechanism, in which only values that are repeated throughout the training data are represented in the final model. Less commonly occurring patterns caused by noise therefore have less impact on the model's classification output.

L_1 regularisation introduces a penalty to the loss function by determining the sum of the absolute magnitudes of weight values:

$$\Omega(w) = ||w||_1 = \lambda \sum_{i=0}^{d} |w_i|_1 \tag{2.5}$$

The parameter update when applying this regularisation factor is:

$$w_{new} = w_{old} - \alpha\lambda s - \alpha\frac{\partial L}{\partial w_{old}} \tag{2.6}$$

with s being the partial derivative of $|w_{old}|$, which is -1 or 1 when $w_{old} < 0$ or $w_{old} > 0$, respectively. When w_{old} is exactly 0, s can simply be set to 0. The main difference of L_1 regularisation in comparison to L_2 regularisation is that the former influences new weights in a linear manner, thus allowing for weights to become 0 more easily than L_2 regularisation does. When applied to a layer in the neural network, L_1 regularisation forces certain inputs or connections to be dropped, which thus allows for feature selection or sparse networks [11, 15, 16].

Layer regularisation

The regularisation methods mentioned above can be applied to different parts of a network's layer, subsequently acting as kernel regulariser, bias regulariser, or activity regulariser [17]:

- **Kernel regulariser**
  Regulariser applied to the weights of fully connected layers of the neural network. Bigger weights are penalised, while the bias remains unchanged.

- **Bias regulariser**
  The bias regulariser forces biases between layers to be small.

- **Activity regulariser**
  Regulariser operating on neuron activations, ultimately affecting the output of a layer. As a result, excessive reliance on specific neurons is discouraged.

Additionally, *dropout* is often employed in neural networks to reduce their complexity. Dropout allows for the removal of a subset of nodes from input layers as well as hidden layers during a training instance, resulting in setting the corresponding input or hidden units to 0. It is therefore similar to the addition of noise in the form of data masking. By forcing the network to update weights with a varying combinations of neurons, more features are considered during training. This introduces redundancy among input features, thereby discouraging overfitting [11, 15]. A recent study by Liu et al. [18] has shown that dropout not only helps against overfitting, but mitigates underfitting, as well. The authors state that by implementing dropout in different stages in the training process, randomness in data is reduced, causing hidden patterns to be captured with less effort.

2.1.4 Performance quantification

A large amount of different variables influence a neural network's classification performance, such as weight values, amount and type of input data, hyperparameters etc. A suitable performance metric is necessary to train, evaluate and compare neural networks and their classification results. A naive approach would be to rely on a model's accuracy, which is simply the proportion of correctly classified data points out of all data points. This metric could result in an unrealistic performance for class-imbalanced datasets. Therefore, in this work performance is quantified by the F1-score, which combines the metrics precision and recall.

Precision represents the proportion of true positives out of the total classified positives. It thus gives an insight in how many of all classified positives were actually positive. It is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{2.7}$$

with $TP = True\ Positive$ and $FP = False\ Positive$. It is an important measure when the cost of a false positive is high.

Recall represents the proportion of true positives out of the total actual positives. It provides information on how many of the actual positives were captured. It is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{2.8}$$

with $FN = False\ Negative$. This measure is important when the cost of a false negative is high. These metrics are summarised in the confusion matrix in Table 2.1 below.

Table 2.1: Confusion matrix.

	Actual Positive	**Actual Negative**	
Classified Positive	True Positive	False Positive	*Total classified positives*
Classified Negative	False Negative	True Negative	
	Total actual positives		

For the classification of activities both precision and recall are favoured to be as high as possible. A combination of the two is therefore used for quantifying the model's performance by means of the F1-score. This score ranges between 0 and 1, representing a balance between precision and recall by computing the harmonic mean of the two, as defined as follows [19, 20]:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.9}$$

The F1-score is used in this work to evaluate optimised models with, as it indicates properly whether a model has classified activities correctly even in non-identically and independently distributed (IID) data circumstances.

2.2 Federated Learning

This section explains the concepts involved in FL. First, the procedure of an FL framework as well as the differences between centralised and decentralised FL are described. Then, the relevant features of the FL methods are highlighted. Lastly, the use of CSI for HAR is briefly elaborated upon.

2.2.1 Federated learning framework

The training process of a model using FL involves five steps, which are iteratively repeated until convergence is achieved or for a predefined number of training rounds [21]. The pseudocode of the FL process is given in Algorithm 2, with FL training rounds R, local epochs E, client set C, and batch size B

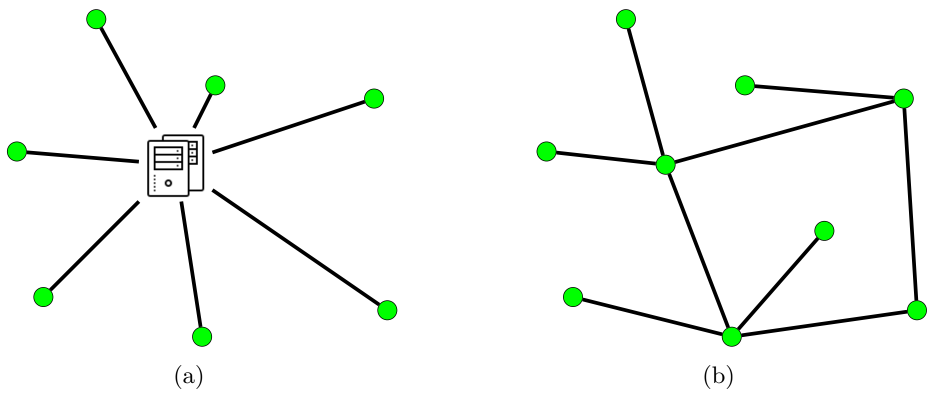


Figure 2.2: Two different topologies with (a) a star-topology for centralised FL with nodes (green) having a communication link (black) to the central server, and (b) a decentralised topology with nodes having communication links to other nodes. Nodes in these topologies can be any IoT device, microcontroller or smartphone capable of recording CSI data.

1. **Client selection**
   A set of clients is selected based on specified requirements, such as data availability, power level, scheduled tasks, etc.

2. **Model sharing**
   The current model is sent from the server to all selected clients. The topology of the central server FL method is a star topology, in which each client is connected to the server via a communication channel (see Fig. 2.2).

3. **Client update**
   Each client that received the updated model now computes updates to the model weights using locally available data for a certain number of epochs. Alternatively, only the gradient can be computed. In that case, the central server updates the global model in a later step based on all gradient updates, instead of updating the global model based on all updated weights. The latter has been shown to lead to a faster convergence rate [22].

4. **Aggregation**
   Local model updates are sent back to the central server, where model updates are aggregated via one of many methods (see Table 3.1). Thus, the update represents all clients participating in the current FL round.

5. **Model update**
   The aggregated update is applied to the global model.

The Federated Averaging method (`FedAvg`) as proposed by [23] is one of the most commonly used aggregation algorithms for FL [21]. Clients perform multiple SGD steps on the loss function locally, then transmitting the updated local model to the central server. Weights are then aggregated by simply determining the weighted average of all weights, proportional to the number of data samples of a client.

Algorithm 2 Federated learning algorithm.

```
1: procedure FederatedLearning($R, E, C, B$)
2:     Initialise global model parameters
3:     for $r \leftarrow 1$ to $R$ do
4:         Select clients $C$
5:         for $c \leftarrow 1$ to $C$ do
6:             Share model to $c$
7:             for $e \leftarrow 1$ to $E$ do
8:                 for $b \leftarrow 1$ to $\frac{\text{length}(\text{LocalData}_c)}{B}$ do
9:                     Select batch $\text{LocalData}_{c,b}$ from client $c$ of size $B$
10:                    Perform forward propagation on $\text{LocalData}_{c,b}$
11:                    Compute local loss and update weights
12:                end for
13:            end for
14:            Send local weights to server
15:        end for
16:        Aggregate weights from all clients
17:        Update global model parameters using the aggregated weights
18:    end for
19:    return Trained federated model
20: end procedure
```

In contrast to the centralised FL setting, methods exist that do not rely on a central server for aggregation of the model updates. In a decentralised FL setup, no central server is present. Instead, clients communicate among themselves to reach model consensus, and thus communication to the server is replaced by peer-to-peer communication [21, 24]. A central authority may still exist [24], e.g. to determine the model to be trained, algorithm to use, setting hyperparameters, performing debugging, etc. A comparison of the topologies of a centralised and decentralised setting is depicted in Fig. 2.2. Due to the absence of a central server, clients do not have to communicate with the same entity. This results in a topology where clients are represented as nodes and communication between two clients is represented by an edge between nodes. Communication does not have to go through all nodes, as is necessary in the star topology, resulting in reduced communication congestion [25]. An additional advantage of the decentralised FL topology is not having a single point of failure: nodes participating in the system can drop out without blocking the entire learning process. In general, in a setting with low bandwidth or high latency, decentralised methods outperform centralised ones [26].

A basic method for decentralised FL is decentralised SGD (`D-SGD`)[27]. Clients perform a single local update step, after which a single communication step between clients follows. Each client then aggregates its own updated parameters with those received. The local update step follows again, etc. Through consensus, the network converges with an accuracy higher than `FedAvg`, but at the cost of a longer wall-clock time. Methods from centralised FL such as `FedAvg` could be implemented in a decentralised manner, starting with the decision of a master node that performs the aggregation of model updates. However, single point of failure issues then still remain, as well as having to deal with bottleneck problems in case of increased communication to the chosen master node.

FL methods can be roughly categorised by four features and their relative implementation: non-IID data, personalisation, communication reduction, and data compression. The features are briefly explained in the following.

Non-IID data

The data used for training are ideally IID, but in realistic settings the data are often non-IID, which is caused when participating clients are only available in a specific time window, have specific behavioural patterns, or are located in specific locations. This could result in problems like client drift, which is characterised by a dissimilarity between the client gradients and the converging global model. An important aspect of an FL framework is being able to converge in a tolerable amount of time in the presence of non-IID data. However, in general `FedAvg` works with non-IID data and thus convergence is not guaranteed [28].

Data can be non-IID in multiple ways, as summarised below. Distributions are compared between different clients. The same naming convention as adopted by Kairouz et al. [21] is used:

- **Feature distribution skew:**
  Differently distributed features.
- **Label distribution skew:**
  Differently distributed labels.
- **Quantity skew:**
  Different amount of data.
- **Same label, different features:**
  The same label results in different features for different clients due to environmental conditions.
- **Same features, different label:**
  The same features result in a different label for different clients due to personal preferences.

Personalisation

Personalised models aim to recognise personal characteristics after updating a global model using local data. Models are first trained globally using features of all devices, after which each device updates the model with local data before using the model. The result is a model adapted to the specific circumstances of each client, without the need for a client to train the entire model independently. The number of local updates can vary between different personalisation methods. The algorithm `per-FedAvg` as described in Appendix C is specifically designed to be able to require only a few update steps [8].

Communication reduction & compression

In this work, *communication reduction* and *compression* features are considered to only cover those methods that actively reduce communication or implement data compression techniques, respectively. For communication reduction, this means that the method is designed to consume less communication bandwidth and does not ignore this aspect. For methods including compression techniques, this means that the method incorporates data compression on data that is sent to other nodes. Incorporating data compression leads to reduced communication within the network. In most cases, decentralised networks require less communication than the centralised equivalent, since there is no central server that needs to to communicate with each client. Therefore, the decentralised FL setups experience reduced communication.

2.3 Wireless Human Activity Recognition

Both the Received Signal Strength Indicator (RSSI) and CSI are readily available metrics suitable for wireless HAR in Wi-Fi interfaces that are commercially available. Therefore, devices that are already present in certain environments could be used to perform HAR. Other methods mentioned in literature [3, 4] are based on Universal Software Radio Peripheral (USRP) devices, e.g. using Frequency Modulated Carrier Wave (FMCW), by which the frequency shift in a signal caused by human interruption can be measured. However, such methods require customised hardware and thus are less suitable for use on larger scale than commercially available off-the-shelf devices. Comparing RSSI and CSI, the latter is able to achieve a higher activity recognition accuracy, since it contains more information and has a higher information resolution. This is because CSI data are based on 30 to 265 subcarriers, thus containing the same amount of complex numbers which vary enough to classify between different activities [3].

The definition of CSI is now formulated based on the description presented in Chen et al. [29]. CSI data contain information on the amplitude and phase of a wireless signal. The two main principles used to model CSI in Wi-Fi signals are multiple inputs multiple outputs (MIMO) and Orthogonal Frequency Division Multiplexing (OFDM). Multiple subcarriers are used to transmit data over different frequencies, as represented by received signal $\mathbf{y}_i$ for subcarrier i [30]:

$$\mathbf{y}_i = \mathbf{H}_i \mathbf{x}_i + \boldsymbol{\nu} \quad \text{for} \quad i = 1, 2, ..., m \tag{2.10}$$

with the CSI channel state $\mathbf{H}_i$, the transmitted signal $\mathbf{x}_i$, noise $\boldsymbol{\nu}$, and total number of subcarriers m. A single CSI measurement is then expressed by a three-dimensional $N_t \times N_r \times m$ matrix $\mathbf{H}$ (see Fig. 2.3), with N_t the number of transmitters and N_r the number of receivers, which represents the signal received from all multi-paths [30, 31]. The entries in $\mathbf{H}$ consist of real and imaginary components [5], and can be used after preprocessing as input to a deep learning model to classify human activities [32].

A relevant constraint for HAR is the minimal update rate of the devices, which should be high enough for performing HAR. Human movement usually lies between 0 and 20 Hz, with daily activities between 0.3 and 3.5 Hz [33]. Additionally, voluntary human movement generally does not exceed a frequency of 10 Hz. Applying the Nyquist rate would require the devices used in HAR to be able to perform updates at a rate of 20 Hz to properly monitor a subject's activity [34].

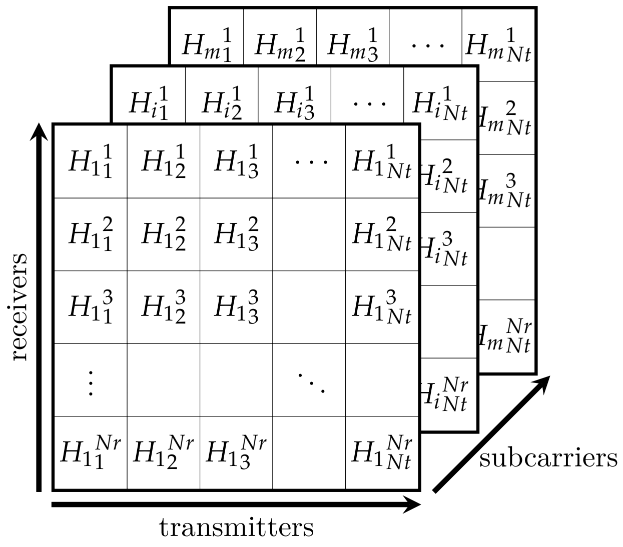


Figure 2.3: Visualisation of matrix **H**.

Chapter 3

State of the Art

This chapter goes into detail about current developments of FL, focusing on the relevance of it in HAR. An overview of methods for FL is presented for both centralised and decentralised settings, each method having distinctive features. The chapter ends with an overview of open challenges for HAR using FL.

3.1 Comparison of Federated Learning Methods

Various methods tackle problems caused by non-IID data, while also certain implementations focus on dealing with reduction of communication or adding data compression. A selection of the most important averaging algorithms is presented in Table 3.1. The overview in this table consists of 24 different algorithms, each categorised by FL features as mentioned in Section 2.2. For every algorithm the table notes whether it operates in a centralised or decentralised setting, whether it is robust against non-IID data, implements personalised models, communication reduction and/or data compression. The last two columns present the accuracy of test data in case the method operates properly under non-IID data and the dataset on which this accuracy has been tested. The accuracy of methods dealing with IID data is incomparable to the accuracy of methods dealing with non-IID data and is therefore left out. A selection of the more exclusive methods is described in more detail in Appendix C. Summaries of the methods with publication details of the study can be found in Appendix A.

3.2 Developments for Human Activity Recognition

Prior work has shown the possibility of using FL for HAR. Sozinov et al. [6] implement FL to train a model on sensor data from smartphones and smartwatches to perform HAR with an accuracy that matches those in centralised learning settings. An FL approach on HAR using CSI is described in Hernandez et al. [5]. The framework, named *WiFederated*, aims to achieve high prediction accuracy while allowing nodes to train with as little data as possible in an indoor environment, with the possibility of classifying activities in unseen locations. It does so by letting the selected clients $\hat{L}$ from total client set L perform multiple local updates and combining the results using `FedAvg` at the central server. The server can then perform an additional client selection from the set $\hat{L}$ to only consider model updates from the set $\hat{L}'$ to avoid the global model from diverging. Of added benefit is that the model can be trained without the need for a GPU, which is useful when edge devices are used, which often do not contain much computational power. Nevertheless, the model should be trained quickly to allow for new nodes and enable accurate performance within a short time frame. Additionally, since edge devices can change locations, they need to adapt rapidly to the new environment. *WiFederated* handles this by having newly joining nodes personalise the global model with local updates to maximise accuracy for specific locations. Both this work and the work done by Sozinov et al. mention the positive impact of personalising a model with only few local updates. This highlights the importance of performing local update steps in HAR with FL. An overview of these studies and other related work focusing on HAR using CSI data and/or using FL methods is presented in Table 3.2. An expanded overview can be found in Appendix B.

A drawback of *WiFederated* is the use of fixed transmitters and receivers. With such a setting, CSI data are always determined in the same direction and between the same two nodes. As a result, it is argued that an object close to the transmitter might be more difficult to track when at the same time there is an object close to the receiver. This effect is caused by the fact that a signal is affected by its surroundings and experiences fading before arriving at the receiver. Therefore, when an object a is located close to the transmitter, fading causes the signal modification caused by a to be less defined at the receiving node. When there is also an object b close to the receiver, the signal change could be entirely dominated by the spatial impact of object b. This effect grows with an increase in trackable objects due to the increase in complexity of the resulting CSI. ML techniques might still be able to distinguish between activities, but an increase in accuracy is expected when changing the transmitting node in a rotating manner, thus allowing for CSI data to be measured between different sets of nodes and in different directions. Then, also activities performed by objects that are not conveniently situated between two nodes have a higher chance of being correctly classified. Having a decentralised FL setup then is a straightforward choice, allowing for a stand-alone topology that handles both the rotational system and the FL algorithm without the need for a central server. This would also avoid having a bottleneck at the central server when the system is extended with many clients.

3.2.1 Challenges

Over the years research has been conducted considering distributed and federated learning. While many methods improve on existing ones, open challenges still remain: specific applications require fine-tuning existing method and in general the accuracy from methods is relatively low (see Table 3.1). Moreover, indoor HAR relies on devices that are connected to a wireless network. Possible devices include network routers, smartphones, laptops and IoT devices. The use of edge devices in an FL setting introduces challenges regarding the available resources [60, 61]. A selection of the challenges in FL is made, specifically focusing on applying FL in HAR with the use of CSI data.

Table 3.1: Comparison of FL algorithms.

Method	**DFL/FL**	**Non-IID**	**Personalised models**	**Communication reduction**	**Compression**	**Accuracy**	**Dataset**[a]
Adaptive FL [35]	FL	✓	-	✓	-	ca. 0,51	CIFAR-10
CHARLES [36]	FL	✓	-	✓	-	0,86	MNIST
FedAvg [23]	FL	-	-	-	-	-	-
FedCS [37]	FL	✓	-	✓	-	0,54	CIFAR-10
FedDL [7]	FL	✓	✓	✓	-	0,95	Self-collected LiDAR data
FedProx [38]	FL	✓	-	-	-	0,65[a]	Synthetic, MNIST, FEMNIST, Shakespeare, Sent140
In-network computation [2]	FL	?[b]	-	✓	-	-	-
It STC [39]	FL	✓	-	✓	✓	0,8	CIFAR-10
Per-FedAvg [8]	FL	✓	✓	-	-	0,71	CIFAR-10
q-FedAvg [40]	FL	✓	-	-	-	0,65-0.85[c]	Synthetic, Vehicle, Sent140, Shakespeare
SCAFFOLD [41]	FL	✓	-	-	-	0,8	EMNIST
Mime(Lite) [42]	FL	✓	-	✓	-	0,86	EMNIST
Structure and sketched updates [43]	FL	-	-	✓	✓	-	-
BLADE-FL [44]	DFL	✓	-	✓	-	0,6	CIFAR-10
Blockchained FL [45]	DFL	-	-	✓	-	-	-
C-DFL [24]	DFL	✓	-	✓	✓	0,8	CIFAR-10
Combo [25]	DFL	?[b]	-	✓	-	-	-
D-cliques [9]	DFL	✓	-	✓	-	0,6-0,8[d]	CIFAR-10
DeLi-CoCo [46]	DFL	-	-	✓	✓	-	-
D-SGD [27]	DFL	-	-	✓	-	-	-
LD-SGD [47]	DFL	✓	-	✓	-	0,7	CIFAR-10
MATCHA [48]	DFL	-	-	✓	-	-	-
PD-SGD [49]	DFL	-	-	✓	-	-	-
TT-HF [50]	Both	✓	-	✓		0,7	Fashion-MNIST

[a] Accuracy tested with open-access datasets: CIFAR-10 [51], MNIST [52], EMNIST [53], Fashion-MNIST [54], Synthetic, Vehicle, Sent140 & Shakespeare [55].
[b] Not mentioned in study.
[c] Depending on the type of data.
[d] Depending on topology, momentum & averaging methods.

Accuracy and convergence rate

There are different sources that negatively affect the convergence rate of a model. Mao et al. [36] list a number of such sources:

- **Channel noise**
  Noisy communication channels can cause loss of information such as model updates.

Table 3.2: Overview of related literature on human activity recognition using FL, CSI or both.

Study	**Recognition type**	**Classifier**	**Data used**
[56]	Activity	SVM	Wi-Fi CSI
[57]	Activity	HMM	Wi-Fi CSI
[58]	Activity	SVM	Sensor data from Google Glass, smartphones and vehicles
[6]	Activity	DNN & softmax regression	Sensor data from smartphones and smartwatches
[5]	Activity	DNN	Wi-Fi CSI
[59]	Gesture	CNN & RNN	Wi-Fi CSI

- **Non-IID data**
  Clients with different data distributions may force the global model to converge slowly, or in the wrong direction.

- **Imperfect CSI**
  The presence of imperfect CSI results in a channel estimation error, which accumulates over time.

As shown in Table 3.1, there are many methods to deal with non-IID data. The presence of noisy communication channels or imperfect CSI is often ignored. However, both have an impact on the convergence rate, and more research into mitigating the impact is necessary to improve the convergence rate under such conditions.

Non-IID data can induce bias due to repeating patterns in participating clients. When clients meet eligibility requirements, it could occur that the model is only trained by clients in similar circumstances. For example, a requirement for a client to participate in computing a model update can be to be idle and being charged. However, it is likely that devices exist that are always in use when being charged. If these devices have access to differently structured data due to them being used in different environments, such data might never be captured in the global model. Yet, the global model is eventually also deployed on such devices. There thus exists bias towards certain devices which could keep the model from generalising properly, causing unfairness in the model [21, 62]. This open challenge is to be researched more before FL can be used on a greater scale. Research could be focused on relaxing eligibility requirements, personalisation of the global model [7, 8], or introducing algorithms that aim towards higher fairness [40, 63].

Another impact on the convergence rate is caused by momentum. Momentum is a technique applied in ML that allows faster convergence by performing update steps to the weights based on the gradient and the previous update to the weights. Equation 2.2 then becomes:

$$w_{new} = w_{old} - \left(\alpha \frac{\partial L}{\partial w_{old}} + \beta \Delta w_{-1} \right) \tag{3.1}$$

with momentum parameter β determining the amount of influence of the previous update, and Δw_{-1} the weight change of the previous update step [64]. In an FL setting, clients could compute the momentum based on locally available data. Although the idea is similar to the conventional ML, in FL it is difficult to prove the added benefit of momentum to the convergence rate [21]. Especially in non-IID settings or with a low number of clients using small batches during training, momentum can slow down convergence or even keep a model from converging [9, 39]. The use of the above mentioned `D-cliques` can mitigate the issue and ensure momentum is of added benefit [9]. Additionally, `MimeLite` introduces the use of momentum based on the updated server state, allowing for locally applied momentum aiding in approaching the optimal solution faster than `FedAvg` [42].

In addition to the open challenges for training the model, the implementation of the trained network also introduces challenges. When all clients run the same trained model and have access to the same data, it is a straightforward task to classify newly seen data, e.g. in a relatively small environment where multiple nodes are able to observe the same activities. In contrast, when nodes run different personalised models and observe the same activity, there exist a possibility the models' decision differs among the clients in a network, since the nodes may have had (slightly) different types or amounts of training data. In an HAR setting where different nodes monitor the same room, it is an open challenge how consensus over the observed activity can be reached.

Communication

Possible room for improvement lies in reducing the amount of communication between server and client, or between different clients. This issue can be handled by either ensuring that less communication is necessary on communication channels, compression of the data being sent, or a combination of the two. It is especially important in decentralised settings to reduce communication, since nodes are completely dependent on communication with other nodes. Without a central authority managing this, it is important to be as efficient as possible without congesting the communication channels. This could become a larger problem with a growing number of participating clients, as bandwidth shortages might arise [60]. Challenges therefore lie in reducing communication costs while maintaining proper convergence rates.

While methods implement techniques to actively reduce communication, often straggler issues still remain as frameworks regularly require all clients to participate. This may lead to convergence problems, since straggler nodes are constantly either dropped or waited for. It is a challenge to schedule nodes efficiently with stragglers among them, especially when most nodes are straggling nodes [60]. The straggler problem is also a matter of concern in decentralised settings: nodes that rely on straggling neighbouring nodes to average their model must either wait a long time on a straggling node or find a new neighbour. To add to this, clients using gossiping algorithms in a decentralised setting are often not able to communicate on-demand and need fixed neighbours [27]. However, in a constantly changing environment it is desirable to be able to connect to different clients, e.g. when a node has no battery left, experiences low bandwidth or moves away from the network. A flexible network topology with changing communication channels should be investigated. For example, the `Combo` system [25] implements an approach in a decentralised setting where pull requests to a node are simply sent to other clients when said node is unreachable, thus making advantage of the fact that a client does not have fixed neighbours which it has to rely on. Other solutions to reduce communication include reducing the amount of data to be sent to other nodes or scheduling nodes to only transmit when data can be received by the receiving nodes.

Simply reducing communication is not always the right solution. For example, FL compared to traditional deep learning requires less communication and computation, while resulting in lower accuracy [6]. This shows that setting the right value for variable parameters have to be chosen carefully to maximise the advantage of FL. For example, choices can be made regarding averaging algorithms, communication methods, compression techniques and network topologies. How such choices influence the network convergence and accuracy is to be investigated.

Data compression is already widely implemented in several studies [24, 39, 43, 46, 65–68]. These studies show that it is possible to reduce the bandwidth necessary to reach a converged model while maintaining a high prediction accuracy. Other studies mention that data compression could be of added benefit to their work, but have not tested this empirically [6, 37, 48, 49, 69]. It is recommended to experiment with different setups and determine how data compression helps reduce communication. As a starting point, top-k sparsification can be investigated, as [39] mentions that it suffers less from non-IID data than other compression methods.

With all methods that reduce the amount of communication or implement compression techniques a trade-off can be observed with the computational capabilities of the framework, as highlighted for certain methods in Appendix D. Using more clients in either a centralised or decentralised setting increases the amount of communication, but total computational power increases, resulting in a higher convergence rate. For better understanding in whether an aggregation or communication framework is beneficial, it should be investigated if the framework results in faster convergence time while communication costs stay within acceptable limits. For example, Konstantinos et al. [70] study the effect of increasing communication on convergence rate in a consensus-based decentralised setting. It concludes that introducing less communication over time can lead to faster convergence. For specific applications of FL such as HAR, it should be validated empirically whether setups with less communication such as decentralised networks perform better than centralised ones, preferably by comparing the two.

Limited resources

In an FL setting with clients that have limited access to power, computational resources, communication bandwidth, and memory, it is a challenge to maintain a model which converges fast enough, yet fits within these constraints. Communication reduction has been discussed before, but dealing with limited power and computation is just as important. One possible approach is to adapt the amount of work done by clients based on available resources such as battery level or available memory [37]. It is necessary to verify whether this method works in a decentralised setting. A different novel approach is to maximise the number of participating clients based on the latency of a client and the available bandwidth [71]. In general, more participating clients results in a faster convergence time, thus maximising the number of clients while respecting system constraints ensures the best possible convergence time.

Dealing with memory constraints can be difficult with a complex model. Using model compression techniques can then help towards a memory-efficient model. Existing software libraries can be used to quickly realise compressed models, such as TensorFlow Lite, which can be used to store a model in the memory of a microcontroller [5].

Privacy

One of the main reasons for the significance of FL is the isolation of data on local devices. This prevents clients from having to share personal data with a server, which is relevant in HAR due to the data containing information on someone's private space. While FL improves on privacy concerns compared to the classical ML setting (i.e. centralised, non-distributed learning), it is worth noting that privacy issues still remain. Some existing methods improve on convergence time, but require sharing a small amount of client-specific data thus compromising on privacy. Other issues are noted below.

- **Malicious Actors**
  Geiping et al. [72] note that in a setting where gradients are shared by clients, it is possible to reconstruct the original data based on the information the gradients carry. This work was carried out on image classification networks to reconstruct images. It should be noted that the data of an image are closely related and that as a result image classification networks are more vulnerable to such an attack. However, the work concludes by stating that the successful attack indicates the rise of stronger attacks to other applications in FL.

- **Adversarial server**
  As noted earlier, a central server is a single point of failure. This also means that clients have to trust the server, both its capabilities and its integrity. An adversarial server can become a problem when compromising communication channels or when actively sabotaging the system, e.g. through a Sybil attack.

- **Possible solutions**
  Wahab et al. [62] note possible solutions that should be focused on in the future, which include noise injection and the ability to adapt privacy preserving methods to the specific needs of a clients. This could differ based on the type of data, model settings, communication methods, and device specifications. An additional approach is to encrypt data sent to the central server or within clients.

Chapter 4

Pilot Study

Before conducting the main experiment to gather realistic data on ADL, a pilot study is performed using a single transceiver pair. This study helps answering questions about the data and classification methods: Is the measured data suitable to perform ML on and are the chosen activities different enough to accurately classify the type of activity? What type of neural network should be used to correctly classify the data? What happens to the classification performance when noise is introduced in the form of a second person performing activities? A network learning to recognise indoor activities might be influenced by the location of the executed activity relative to the location of both the transmitter and the receiver, as hypothesised in Chapter 1. This study helps answering these questions and aims to indicate important aspects in HAR to establish the focus for subsequent experimental setups with more transceiver pairs. Additionally, the data gathered during this experiment are used to tune the hyperparameters of the neural network.

This chapter describes the experimental setup of the pilot study, the data collection process, the data structure, and an analysis of the impact of several environmental factors on the classification accuracy of HAR.

4.1 Setup

The pilot experiment is performed in the break room of the research group. For the measurement of CSI data, two identical devices are used as nodes with a communication channel between them. The nodes n_0 and n_1 are placed at either end of the room on a table at the same height. Any object obstructing the line-of-sight between the two nodes is removed. A subject executes activities close to node n_0 in area p during each CSI measurement, while another subject introduces noise into the environment by means of random movement in area q. A schematic overview of the experiment setup is presented in Figure 4.1.

4.1.1 CSI measurement

CSI data are measured between node n_0 and n_1 while executing one of the activities A_i, with $i \in$ [sit/stand, eat/drink, work, walk, nothing]. The activities are presented in Table

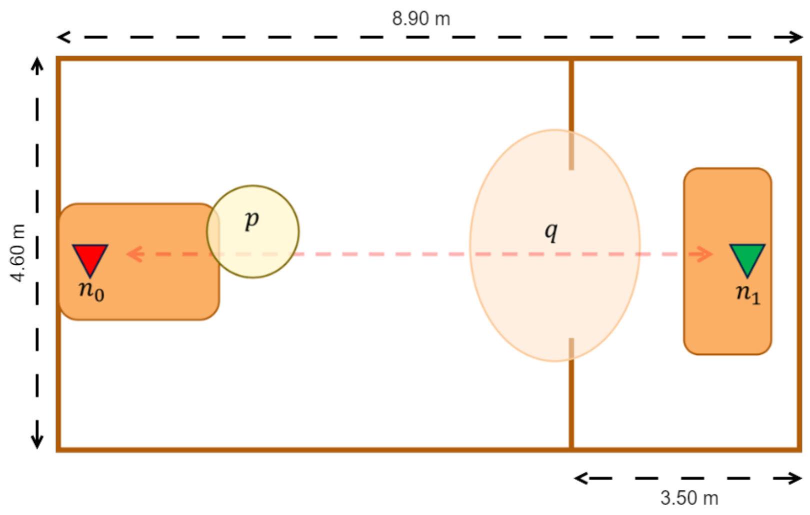


Figure 4.1: Setup experiment 1, with CSI channel (red line) in between node n_0 and n_1. Activities are executed in area p, interference is performed in area q.

4.1 with a brief description. All activities are of relevance in an indoor environment suitable for living in, as they are either ADLs, or other activities one might perform inside their home.

Table 4.1: List of activities for experiment 1 with corresponding description.

Activity	**Description**
Sit/stand	Sit in a chair and stand up. Repeat at comfortable speed.
Eat/drink	Eat and drink or pretend to do so.
Work	Work behind own personal device.
Walk	Walking around in activity area
Nothing	No presence or activity in the room

The activities are performed by two test subjects, each subject executing all activities both with and without the noise as generated by random movement by the second test subject. Each activity-noise combination is repeated twice, in which both nodes once act as transmitter and once as receiver. This allows for the execution of the activity both close to the transmitter and close to the receiver. An overview of the performed CSI measurements for each activity is presented in Table 4.2. A complete experiment schedule is presented in Appendix E.

During each measurement, CSI data are sent with one antenna and received by two antennas, using the Intel Wi-Fi 6E AX211 Module [73]. The PicoScenes platform is used to perform CSI measurements [74]. The receiving node gathers data using the *responder* mode, at a sampling rate of 100 Hz using two antennas for 60 seconds. As highlighted in Section 2.3, this sampling frequency should be high enough to be able to capture all relevant human movement. The data are injected by the transmitting node with the *initiator* mode, with

Table 4.2: All executed scenarios during the pilot experiment. This is done for each measured activity.

Scenario number	**Subject**	**Location**	**Interference**
1	0	RX	✓
2	0	RX	×
3	0	TX	✓
4	0	TX	×
5	1	RX	✓
6	1	RX	×
7	1	TX	✓
8	1	TX	×

random Wi-Fi packets being sent using one antenna, spread over 57 subcarriers with a center frequency of 5.2 GHz. The use of this frequency allows for a greater number non-overlapping Wi-Fi channels compared to the frequently used frequency of 2.4 GHz. Additionally, the shorter wavelength allows for measuring smaller changes in CSI magnitude than a frequency of 2.4 GHz. The $\mathbf{H}$ matrix resulting from the measurements has dimensions $1 \times 2 \times 57$. After each measurement, the data are saved to a `.csi` file with a filename indicating the performed activity and all information that can be found in Table 4.2.

4.2 Data Structure

The data gathered in the pilot study consist of equal amounts of data on each activity, except for the "nothing" class, which was measured over just two measurements (once for each location). The CSI magnitude differs greatly per activity and per measurement. To show this, the magnitude distributions of each class for randomly selected measurements and subcarriers are presented in Figure 4.2. It can be observed that neither class follows a normal distribution, which is relevant when deciding on a data normalisation or standardisation technique. The data are identically distributed across devices, yet not independently, as each activity class is measured by multiple devices simultaneously.

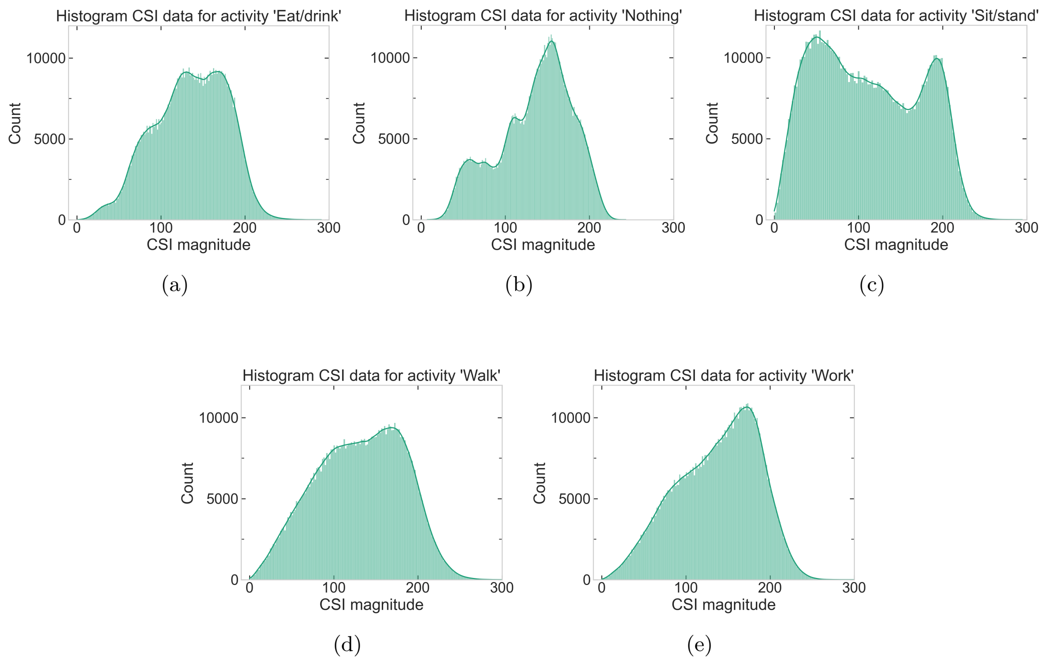


Figure 4.2: Data distributions for activities performed during the pilot experiment.

4.3 Observations

The data are classified using a small neural network consisting of 3 dense layers, inspired by *WiFederated* [5], in order to verify if the network is also suitable to classify the data in this work. The boxplot in Figure 4.3 presents a comparative overview of the model performances under different environmental factors. Difference in interference is presented by scenarios with and without interference. A difference in distance to the measuring node is indicated by whether the activity is executed close to the receiver or the transmitter, and personal differences are shown by a varying test subject.

As observed, the activities executed without interference are consistently more accurately classified compared to those with interference. This can be intuitively explained by the fact that interference introduces noise into the measured data, thus increasing its complexity.

Secondly, F1-scores are influenced by whether an activity is executed close to the receiver or close to the transmitter. The first quarter of the graph is compared to the second, and the third to the fourth. In these comparisons, all environmental aspects are the same except for the location of the activity. It is observed that for each of the compared quarters, the same scenario scores consistently higher when an activity is performed close the transmitting node, in contrast to what was hypothesised in Section 3.2. It is argued that this occurs because of multipath signals, which is caused by signals being reflected and scattered throughout the environment [75]. When such a signal arrives at the receiver, the signal has lost a portion of its amplitude due to downfading. An activity performed at that location only influences the signal minorly compared to doing so close to the transmitter, where signals

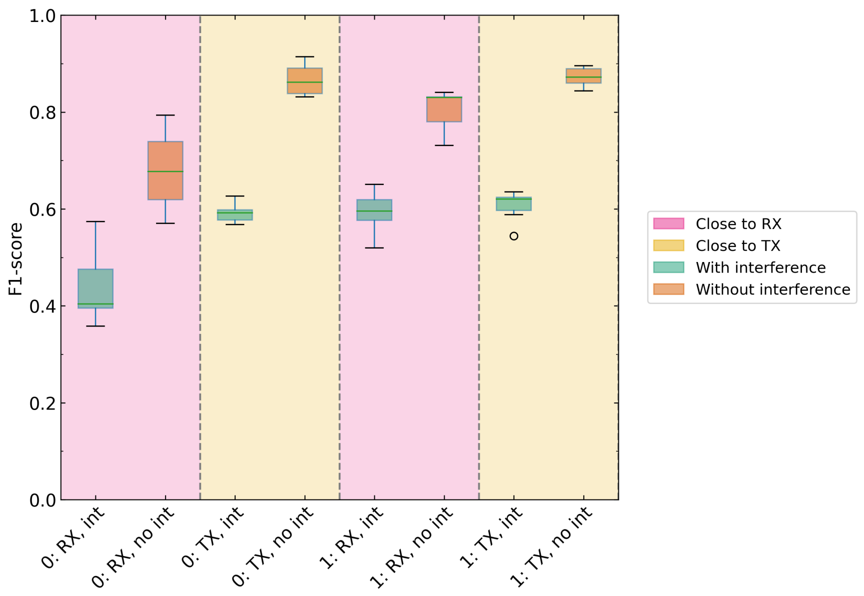


Figure 4.3: F1-scores of different scenarios. Activities are executed by subject `0` or `1`, close to receiver (`RX`) or transmitter (`TX`), with interference (`int`) or without interference (`no int`). Matching variables are marked by the same colour.

still have their full strength. The difference in F1-score indicates the importance of varying the transmitting direction in transceiver pairs when activities are performed throughout multiple locations in a room.

Lastly, it is observed that the difference in F1-scores across different locations varies for subject 0 and 1. This difference can be associated with differences in physical attributes, as well as minor variations in the activity execution, resulting in dissimilar CSI data. Moreover, even execution of activities by the same individual may result in varying measurements. These findings highlight the fact that CSI data differ for distinct individuals, presenting a challenge for developing a universal classification model in dynamic environments. Considering this observation, a large number of subjects are invited to participate and perform activities in the main experiment.

Chapter 5

Data Gathering & Preprocessing

5.1 eHealth House Experiment

The pilot study was performed to fine-tune hyperparameters and to decide on specific settings for further experimental setups. This experiment only involved two communicating nodes and did not consider the numerous communication channels that would exist in a realistic scenario where IoT devices are scattered across a real-life environment. To simulate a more realistic setting in which CSI-based human activity recognition might be used in the future, data are gathered in the eHealth House. This experimental area similar to an apartment of a two person household is located on the University of Twente and allows for experimentation and observation of activities in a realistic and controlled indoor setting [76].

This experiment aims to provide insights into how activity location, client participation and location, the type of activity, and the presence of more interference affect the accuracy of classification. The data collected from this experiment are also used to simulate different types of communication settings and learning systems to determine the effect of various methods.

5.1.1 Setup

In the eHealth House, participants perform activities in different locations, with nodes sending and receiving CSI data in different locations inside the area. A total of 18 participants are brought into the eHealth House with three participants at a time. Prior to the experiment, all participants signed a consent form indicating their willingness to participate in the study. Each participant is asked to perform an activity in one of three locations: L_{TV}, $L_{kitchen}$ and L_{table}. The nodes used for measuring CSI data are placed in four locations which represent realistic location of actual wireless devices. Figure 5.1 shows a schematic overview of the eHealth House, with node identifiers indicating the location of devices. Table 5.1 lists the identifiers and locations of each node, together with its relevance in a realistic environment.

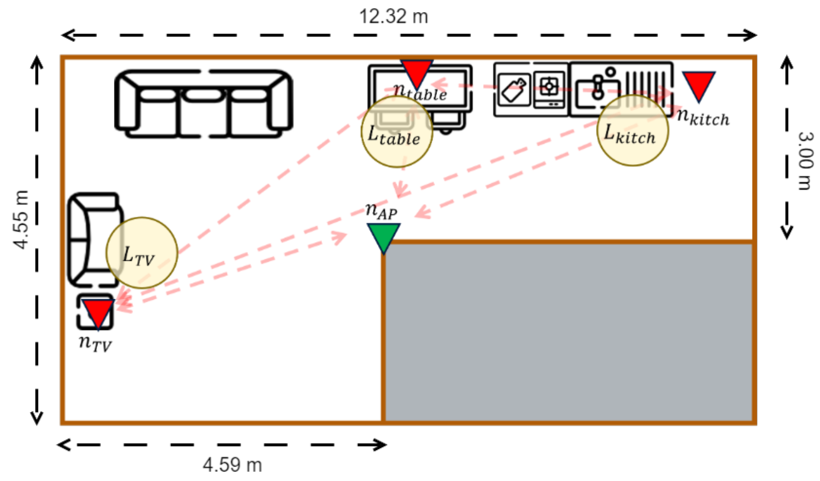


Figure 5.1: Setup experiment 2, with CSI channels (red lines) in between all nodes n_{TV}, n_{table}, $n_{kitchen}$ and n_{AP}. Activities are executed in area L_{TV}, $L_{kitchen}$ and L_{table}.

Table 5.1: Node tags and their corresponding locations, with relevance of presence in indoor environment.

Node tag	**Location**	**Relevance**
n_{TV}	In between the television and the armchair.	Smart TV, home assistant, casting device
n_{table}	On the table between the living room and kitchen.	Laptop, mobile phone
n_{kitch}	On the kitchen countertop, next to the fridge.	Smart fridge or other IoT kitchen device
n_{AP}	In the middle of the eHealth House, elevated (simulating being mounted on the ceiling)	Access point, router

5.1.2 CSI measurements

The experiment involves four activities, as presented in Table 5.2. Detailed experiment schedules can be found in Appendix F. Participants are required to perform one of the four activities, while CSI data are continuously being measured between a transmitter device and three receiver devices for a duration of three minutes. After completing the activity, the same participants execute the next activity at the same location, but with a different node functioning as transmitter. For each group of three participants, one participant performs the same activity four times consecutively. This activity is considered the “main” activity and is the target of the classification. Two other participants perform randomly assigned non-main activities, ensuring that each combination of activities occurs only once for each set of three participants in order to avoid the learning algorithm from recognising activity combinations instead of individual activities. The CSI data resulting from the randomly assigned activities serve as noise introduced into the system. The measurements are repeated until all four nodes located around the eHealth House have transmitted CSI data, and each main activity has been performed four times by one participant. The experiment consists of three sets, with each set involving a different location where the main activity is performed. A total of eighteen different participants take part in the experiment. With three different sets and three participants per set, this means that each set is repeated twice, thus each location hosting the main activity twice.

As opposed to the first experiment setup in which the PicoScenes tool was used together with the Intel Wi-Fi 6E AX211 Module to measure CSI data, for the second set of experiments the Linux CSI Tool [77] was used in combination with the Intel Ultimate Wi-Fi Link 5300 NIC [78], due to a lack of availability of the other devices. For each measurement, CSI data are sent with two antennas and received by three antennas with a sampling rate of 100 Hz. There are 30 subcarriers per measurement, thus the $\mathbf{H}$ matrix has dimensions $2 \times 3 \times 30$. As in the first experiment, data are saved to a `.csi` file with a filename corresponding to the main activity, location at which the main activity is executed, which node functioned as transmitter, and by which node the data are received. The locations and types of the randomly assigned activities are recorded, as well.

Data are labeled by assigning one activity per person per measurement. The labels indicate where the activity is executed and which activity is executed. The data collected from the participants are anonymised and kept confidential, and the experiment is approved by the university’s ethics committee.

Table 5.2: List of activities for experiments in the eHealth House with corresponding description.

Activity	**Description**
Sit/stand	Sit in a chair and stand up. Repeat at comfortable speed.
Eat/drink	Eat and drink or pretend to do so.
Work	Work behind own personal device.
Rest	Sitting idly, while being allowed to look around or change position.

5.2 Data Preprocessing

Various preprocessing steps are applied to the data before being able to use as input for a neural network. First, data are interpolated to ensure a high enough sampling rate, after which data are either normalised or standardised. Then, a rolling average filter is applied to reduce noise. Finally, the data are separated per 100 measurements to ensure each data sample represents a second of CSI data. The different steps are described in more detail in this section.

5.2.1 Interpolation

To achieve a CSI signal of sufficient sampling frequency, the transmitter rate is configured at 100 Hz. However, at the receiver side it might occur that the sampling rate of the received CSI data is not uniform due to packet loss caused by e.g. obstructions in the environment. This results in a sampling rate lower than the desired 100 Hz, from which the learning rate of a neural network can suffer. To guarantee a balanced dataset with sufficient information to allow a network to learn to perform accurate classification, it is necessary to interpolate missing data points [79]. This is achieved by means of linear interpolation, which is defined as follows [80]:

$$y = y_1 + (x - x_1)\frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{5.1}$$

with (x, y) the coordinates of the new data point, and (x_1, y_1) and (x_2, y_2) the coordinates of data point 1 and 2, respectively. Signals with a lower sampling rate than desired can be replicated appropriately by means of interpolation, as long as the sampling rate is equal to the Nyquist rate or higher. For the activities performed in this research, the Nyquist rate is around 5 Hz [81].

5.2.2 Normalisation/standardisation

The data are either normalised or standardised before feeding it to the machine learning model. By doing so, noise is reduced and the stability of the learning process is increased, as well as the convergence rate of the network. One can either apply normalisation or standardisation to the data. The former normalises values between 0 and 1, while the latter scales all values between -1 and 1. Both methods allow weights across a network to be of similar magnitude. If this would not be the case, the loss function and weight updates based on it can become particularly sensitive to features of relatively big magnitude, which affects a network's learning performance negatively.

With normalisation, the Min-Max scaler is applied to each data point, resulting in a value between 0 and 1. This is formally defined as:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{5.2}$$

with x' the updated data point, x the original value of the data point, and $min(x)$ and $max(x)$ the minimum and maximum value of the whole dataset, respectively. An extremely high maximum value can cause biased scaled values, making this method sensitive to outliers [82].

The standardisation or Z-score normalisation method rescales a dataset such that it follows a Gaussian distribution with a mean value μ equal to 0, and a standard deviation σ equal to 1. This method assumes that the data are normally distributed. If not, the distribution of the input data is not preserved in the output. The equation for standardisation is defined as follows:

$$z = \frac{x - \mu}{\sigma} \tag{5.3}$$

Since both the mean value and the standard deviation are influenced by relatively big minima and maxima, this method is not resistant to outliers, either [11, 83].

For simplicity, normalisation methods were limited to the ones mentioned above. After initial observations, data based on normalisation outperformed those based on standardisation methods. Furthermore, Wi-Fi based HAR signals are not normally distributed as apparent from Figure 4.2. Therefore, the Min-Max scaler is adopted in this research.

5.2.3 Rolling average filter

Signal interference and other environmental activity can lead to unwanted noise in the amplitude of the CSI signal. This noise adds to the sudden changes in amplitude already present in CSI data. To mitigate this issue without filtering out important data, a rolling average filter is applied. This method highlights long-term trends while removing random noise. By adjusting the size of the filter window, the amount of data loss can be controlled [84]. This is similar to the filtering approach in *WiFederated* [5]. The rolling average for subcarrier i at current time t can be computed by taking the average of the most recent $S - 1$ samples for a window size S and is given by:

$$\bar{A}_t^i = \frac{1}{S} \sum_{s=0}^{S-1} A_{t-s}^i \tag{5.4}$$

Adjusting the window size S results in different levels of data filtering, with the optimal value determined through hyperparameter tuning.

5.2.4 Federated datasets

A federated dataset consists of data collected by different clients. In this research, such a dataset is implemented by creating an array with a sub-array for each client. The data on each client undergo shuffling to prevent the model from learning correlations between the order of the data and the target labels. By doing so, the model is forced to learn general features that are applicable to any part of the data. The data at the client side are subsequently split into batches of fixed size and each batch is processed separately during training to increase the learning rate. The number of epochs in a client's dataset determines the number of times the model is iterated through by a client during a single federated training round. In a federated dataset, this is achieved by simply repeating the shuffled and batched data x amount of times for x number of epochs [85]. The structure of the data gathered during the eHealth House experiment is presented in Figure 5.2. The federated dataset is subsequently split into a train dataset consisting of 60% of the data, and a separate test dataset of 40% of the data. The sets are used to relatively train and evaluate a neural network with.

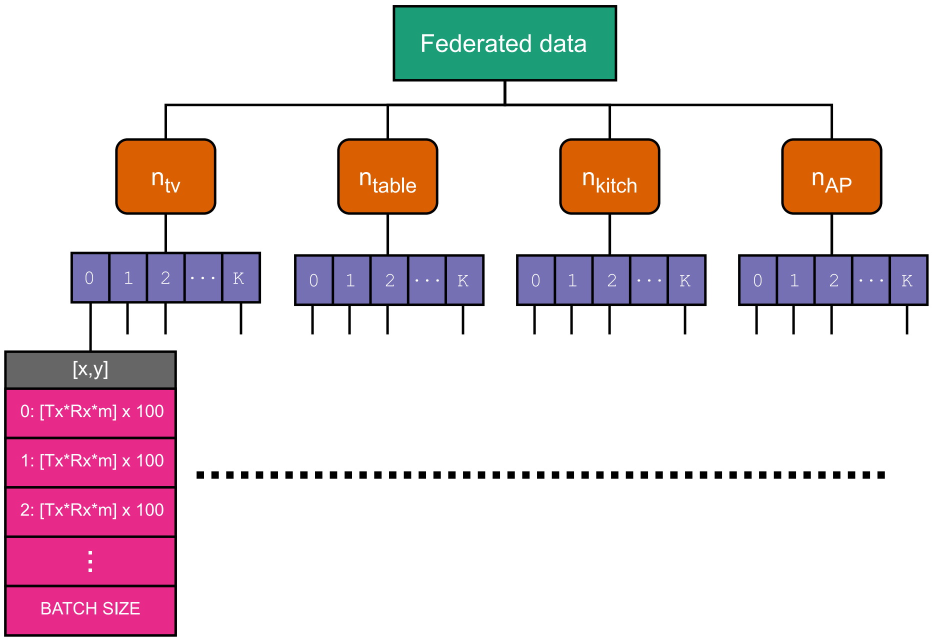


Figure 5.2: Data structure of federated data (green), with data separated per node (orange), with per node up to K entries (purple). Each entry includes input data x with label y (grey) equal to the batch size (pink). All input data are a 2D array with dimensions $(N_t * N_r * m) \times 100$. K is defined as $floor(X/batch\ size)$, with X the total amount of data samples of a node.

Chapter 6

Methodology

This chapter describes the methods as applied in the implementation of learning algorithms, the construction of different datasets, and how the influence of resource constraints is determined.

6.1 Neural Network Model

6.1.1 Architecture

The neural network used in this research is based on *WiFederated* [5], since it has proven to be effective in indoor HAR. The input for the model consists of CSI data with dimensions equal to a flattened $\mathbf{H}$ matrix for each second of input data. When a sampling frequency of 100 Hz is used for measuring data, the input is 2D with dimensions $(N_t * N_r * m) \times 100$. The data are passed through a flatten layer which converts the input to a 1D array, which is defined as the data for all m in N_r, for all N_r in N_t. Subsequently, the array is passed through a dense neural network (DNN) architecture consisting of three fully connected dense layers, each with 100 hidden units with ReLu activation functions. The last layer in the network is another dense layer with a softmax activation function with the amount of outputs equal to the amount of classes in the dataset.

An activity regulariser using L_1 regularisation is implemented on each dense layer of the network, reducing the output of each layer. Next to that, a kernel regulariser using L_2 regularisation is applied, which forces weight values of the dense layers to remain small. The regularisation parameter λ for both regularisers is determined by means of a grid search, in which different λ values are tested and evaluated by recording the F1-score of the resulting network. Additionally, a dropout layer is included between each two dense layers with a rate of 0.5. This means that for each iteration, 50% of randomly selected units is set to 0. During inference, all units are used for the classification. In order to keep the sum of all units unchanged for both inference and training with dropout, all units that are not dropped are scaled by $\frac{1}{1-\text{rate}}$, i.e. the probability of a unit not being dropped. This is known as *the weight scaling inference rule* [16, 86].

The simplicity of the neural network allows for a small model footprint, which is necessary for integration on IoT devices. A visualisation of the model is given in Figure 6.1. A more detailed graph is given in Appendix G.

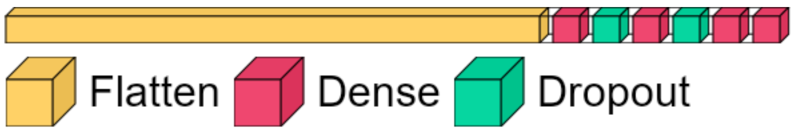


Figure 6.1: Abstract representation of used neural network, with three fully connected dense layers and a softmax dense output layer.

6.1.2 Model implementation

The model described above is implemented using the Keras API in Python [87]. Model optimisation is achieved through minimising the mean squared error (MSE) loss function through SGD as suggested in *WiFederated* and presented below:

$$\mathcal{L}(\theta, x, y) = \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{F}_\theta\left(x_i\right) - y_i\right)^2 \tag{6.1}$$

The learning rate is set to a value of 10^{-5}, and the server learning rate to 1. The loss $\mathcal{L}$ for weights θ, input data x and expected output label y is computed across all possible classes N, in which for each possible output class i the squared difference between the predicted output $\mathcal{F}_\theta\left(x_i\right)$ and the expected output y_i is summed. The input array y is encoded as a one-hot vector, thus each class being represented by either a 0 or a 1 for either not representing that class or representing it, respectively. The loss is a value between 0 and 1 after averaging of all batches, with 1 being the loss value if all predicted outputs are wrong.

The federated implementation of the model is achieved using the TensorFlow Federated (TFF) open-source framework [88]. This framework allows for customisation in the used aggregation method, the number of clients, training rounds, local computation etc. The TFF framework does not allow for activity regularisation yet, thus only L_2 kernel regularisation is implemented for the federated setting.

For network optimisation, a split of 60% of the original data is used, whereas 40% is preserved for testing the model. The categorical accuracy of each training instance is logged, defined by the proportion of correct one-hot labels in a training iteration [89]. After training, the test data are leveraged to determine the F1-score of the classification model.

6.2 Training Datasets

The data gathered during both experiments are separated per receiving node. After going through preprocessing and labelling steps, which include the identification of the type and location of the activity performed, as well as the index of the transmitting node, various datasets can be constructed. The data from the pilot experiment are then simply stored per scenario (i.e. each of the measurements as shown in Table 4.2), after which the data are used to tune hyperparameters and to investigate the influence of environmental differences in the measurement area, as described in Section 6.3. Next to that, combinations of different scenarios are made to optimise the network, after which the models are evaluated on a single scenario. This can help analyse whether models benefit from data measured in varying environments.

The data resulting from the eHealth House experiments are processed to create datasets that each serve a different objective and can be used to determine the influence of various parameters on the performance and convergence speed of a neural network. The datasets are constructed based on either variations in location or differences in receiving nodes. The four resulting datasets are denoted as the *simple*, *complete*, *location* and *client* set, which are briefly described below. Next to that, it is specified how the datasets are used for training models. A summarised overview of the different datasets is presented in Appendix H.

6.2.1 Simple set: single locations

The *simple* set represents a separation in location where the activity is carried out. This means that the data are stored for each of the three locations represented in Figure 5.1, while ensuring the separation of receiving nodes to simulate clients in the FL system with access to only their own data. This set is used to perform parameter tuning of the neural network. To do so, a model as described in Section 6.1 is trained for each location. To determine how certain training settings influence performance, a model is trained for varying regularisation parameters and varying batch size.

6.2.2 Complete set: all locations

The *complete* set is used to answer sub-questions 1 and 2 as stated in Chapter 1, as well as a part of question 5. In the *complete* set, all the data collected in the eHealth House are used, meaning that data collected from various locations during the execution of different activities are all in the set, and data are labelled per main activity. Each client has access to all performed activities, but only what that node has received itself (i.e. locally gathered data). The difference to the *simple* set is that all locations are included in the *complete* set. The performance of a model trained on this dataset serves as a benchmark for comparing the performance of models trained in other scenarios. For example, when a new location is introduced for executing an activity, models trained on this new location can be compared with the benchmark performance of the existing model. The benchmark performance is determined by training a model for varying number of locally performed epochs and number of FL training rounds, using the *complete* set.

As the *complete* set contains all gathered data, it allows for the analysis of the impact of the use of an FL system compared to a traditional machine learning approach, where a single entity performs model optimisation. Next to that, various FL aggregation algorithms are compared to each other using the *complete* set.

6.2.3 Location set: introducing new locations

The *location* set is used to answer sub-question 3 and partly question 5. To simulate realistic environments, individual locations can be excluded from the *location* set, which enables the network to learn from a limited number of locations. Note that this does not mean that nodes receiving the data are removed from the dataset, but only the location where the activity is performed. After a model reaches a predefined *budget*, a new location can be introduced. The budget is defined as the total number of epochs trained per node, i.e. the number of local training epochs multiplied by the number of total FL rounds. Using the model that has already trained on the two old (i.e. already present) locations as starting point, a new model is trained on the new location. This approach aims to enable the new model to reach convergence more quickly as compared to training on a new location from scratch. This becomes particularly relevant when accurate classification of activities in a new location is necessary or when the classification accuracy of activities in a specific location is insufficient. The addition of new locations and re-training on them enables the classification model to quickly adapt to new environments. When it is possible to use an already trained model as starting point for new locations, such re-training could potentially only take few training rounds. This is similar to what is presented in *WiFederated* [5], where it was demonstrated that in FL existing models can be leveraged by performing only a limited number of training repetitions on data from activities of unseen locations.

To achieve re-training on unseen locations, first distinct models are trained for a varying number of locally performed epochs and number of FL training rounds, for each combination of two out of three locations where activities are performed. Comparing the performance of these models allows for an understanding of how different locations affect the classification performance, which is important to decide on how to deal with a change in activity location. Then, the remaining unseen location is fed to the model and training continues based on only the data introduced by the new location, again for different training parameters. The model is trained until a training accuracy of at least 80% is reached, or until a predefined budget of 3000 epochs is exceeded.

6.2.4 Client set: introducing new clients

The *client* set is used to answer sub-question 4 and partly question 5. This set is similar to the *location* set, with the difference being that data are aggregated in such a way that nodes can be conveniently removed during training, instead of activity locations. For each node n_i, separate subsets are created with data transmitted and received by all nodes except node n_i, while other sets are created with only data transmitted and received by node n_i. This allows for simulating scenarios where a new device is introduced into the communication network. A classification model then exists based on data from the nodes that were already present in the system, and the model is to be updated with data sent and received by the new device. This is done to determine whether existing networks can be leveraged to train with data from new nodes, and if so, how fast convergence is achieved.

Similar to the *location* set, training with data from newly added clients is done by first training models for a varying number of locally performed epochs and number of FL training rounds, for each combination of data from three out of four nodes. Removing a node from the training phase is achieved by excluding all data transmitted by said node from the remaining receivers. Additionally, also all data received by said node from the remaining transmitters is removed, such that no data involving the removed node is present. The models trained on the data with the absence of a node are compared to each other to understand how different clients affect model performance. This helps understanding how to deal with a change in client participation. Then, when the node is introduced into the system again, one of the following three schemes is executed:

- The new model is trained locally on the new node, with data received by the new node and transmitted by all other nodes. This represents a device only being able to detect CSI data without being able to transmit any data.
- The new model is trained in a federated manner by the already present nodes, with data received from the new node by all other nodes. This scenario represents a newly added node that can only transmit CSI data and is not able to detect it.
- The new model is trained in a federated manner by all nodes, with data received and transmitted by the newly added node. Here, the newly added node is able to communicate in the same manner as the already present nodes, i.e. transmitting and receiving CSI data.

Again, the new models are trained until a desired training accuracy of 80% has been achieved or when the budget of 3000 epochs is exceeded. Figure 6.2 shows a schematic overview of using the *client* set to train on data from new clients.

It should be noted that the data associated with the new node only consist of data from the past, i.e. when the node was not included into the FL process. Therefore, the methods described above only serve as a proof of concept and represent a real-life environment only up to a certain point.

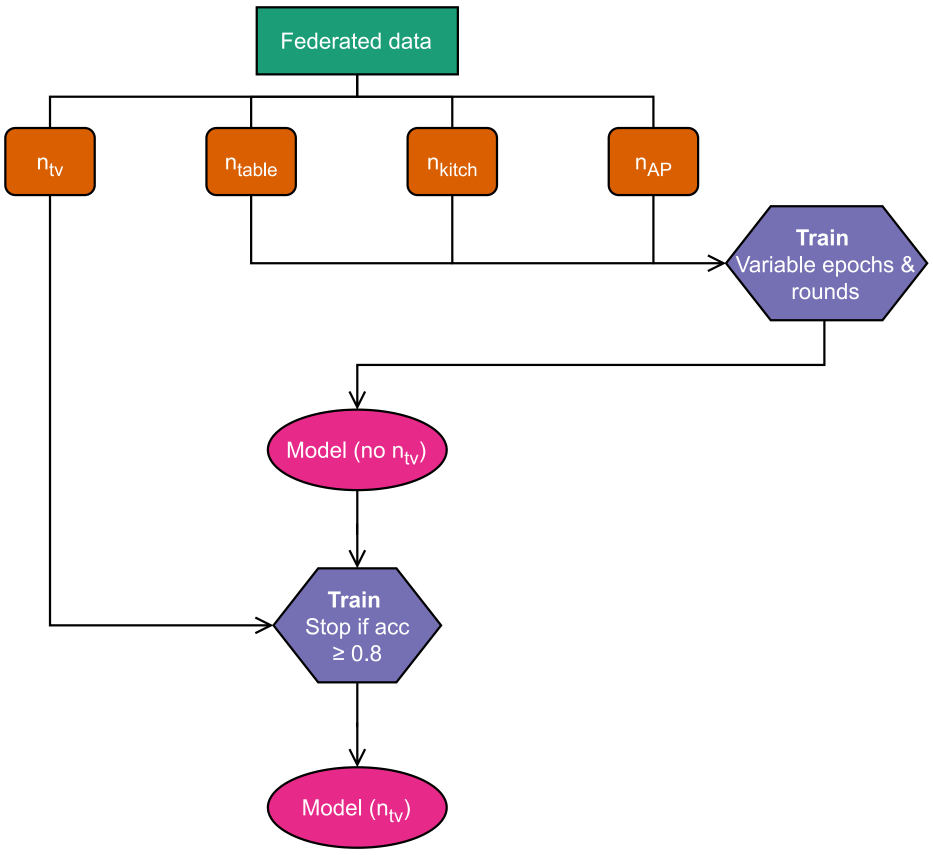


Figure 6.2: Data flow for the *client* set. A model is pre-trained with three of the four available clients for variable number of epochs per round and total rounds. Then, model optimisation is performed with data from the newly added node (n_{tv} here). The result is a model specialised to classify activities based on the new data.

6.3 Influence of Variables

Different variables and parameters are recognised to be influencing classification results in HAR, such as the regularisation parameter λ, available training data and batch size. For the FL system, additional influences are present such as the amount of local computation, the total number of training rounds and the location at which an activity is performed. The different parameters are described briefly below. An overview of all tested parameters is presented in Table 6.1, together with the range of tested values.

Table 6.1: All tested parameters with their corresponding range of values. Not all value combinations are tested.

Variable	**Values**
Batch size	$[10, 20, 40]$
Local epochs	$[1, 5, 10]$
FL training rounds	$[5, 10, 20, 50, 100, 250, 500, 1000, 2000]$
Activity location	$[L_{TV}, L_{table}, L_{kitch}]$
Client	$[n_{TV}, n_{table}, n_{kitchen}, n_{AP}$
Data fraction	$[0.1, 0.2, 0.5, 0.75, 1]$

6.3.1 Parameter tuning

To determine the influence of regularisation parameter λ and to find values that improve classification results compared to having no regularisation entirely, a grid search is proposed over multiple iterations using the network architecture described above, with a 7-fold cross validation during network optimisation. For each next iteration, more ideal values are selected based on the results of a previous iteration. The F1-score for each iteration is recorded, with the aim to select a combination of λ values for both L_1 and L_2 regularisation that achieve the highest F1-score among the compared values.

The batch size used in training the FL system is varied for a subset of the data (in the *simple* set). Not having to train on the complete dataset allows for faster convergence and thus more convenient comparative analysis. Results then might not represent reality to the best extent, however comparisons between models trained on varying batch sizes can still be done.

6.3.2 Machine learning and federated learning

A comparison between classical, non-distributed, centralised ML[1] and FL is made by considering the characteristics and qualities of both implementations based on the theory provided in Chapters 2 and 3. Next to that, the classification performance as well as convergence speed of both methods are compared to each other.

6.3.3 Aggregation algorithm

As indicated in Chapter 3, not every aggregation algorithm is suitable for every FL implementation. Aggregation efficiency can be influenced by aspects such as data distributions or cooperation of clients. In this work, the main influence is how the data are distributed. To analyse how various aggregation algorithms act in FL systems for HAR, three different algorithms are implemented and their performance is evaluated. The algorithms are available through the TensorFlow library and are listed below:

- `FedAvg`
- `FedProx`, with hyperparameter $\mu \in [0.001, 0.01, 0.1, 1]$

[1]Called “classical ML” throughout this work.

- `MimeLite`, with momentum ($\beta = 0.9$, based on [42])
- `MimeLite`, without momentum

Each of the algorithms implement aggregation in a weighted manner, enabling a client with access to more data to contribute more to the global model [88].

6.3.4 Computation and communication

Computation and communication are analysed by varying the amount of local computation and the total training rounds. A training round includes distributing the model from the central server to the connected clients, performing local model updates and subsequently sending the updated model back to the central server. The amount of local computation in FL can be varied by controlling the amount of iterations performed during local model updates. Local training rounds are represented by the number of epochs a client goes through their dataset. A single epoch translates to training on local data once before sending the update to the central server, while n epochs force a client to perform n local update steps. The selected amount of epochs are 1, 5 and 10. These values are based on the desire to explore different amount of local computation, varying enough to clearly determine the impact of doing so while constraining the needed computational resources. More local updates can result in inconveniently high training duration for a single training round.

A higher number of training rounds results in more in-network communication, since the model and its updates are sent back and forth each round. Together with the amount of local computation, it is aimed to determine a trade-off between local computation and in-network communication.

6.3.5 Location and client change

It is analysed how the location of an executed activity in the eHealth House influences classification performance using the *location* set. Next to that, the *client* set allows for convenient removal of a single client from the dataset, which is done so one client at a time. Subsequently, the classification performance of a model trained on data from the remaining three clients is compared for each excluded client. This allows for the analysis of the impact of each individual node on the FL process. Additionally, the introduction of new data gives insights in how to deal with new activity locations and new participating clients.

6.3.6 Limited data availability

The influence of the quantity of available data for training a network is especially relevant when re-training an existing model on new, unseen data, of which the details are explained in Section 6.2. Ideally, such re-training is done as quickly as possible without the need to gather a large amount of data. It is therefore relevant to understand how a lower amount of available data influences network performance. To simulate the situation in which less data are available for a given setup, a varying amount of fractions of the total available data is taken to train the network. For each fraction of data, network performance is determined after training and reaching convergence for both the *location* set and the *client* set.

6.3.7 Resource constraints

To determine the effect caused by limitations in available resources in IoT devices, the following resources are considered, based on the resource constrains as presented by Imteaj et al. [60]:

- **Communication:** The amount of communication between clients and the central server is represented by the number of training rounds, as for each training round the model weights are sent back and forth. An optimised neural network trained through relatively less training rounds thus encountered less communication.
- **Energy consumption:** The energy consumption of a device performing live classification with the trained model is not determined in an empirical manner, yet it is reasoned that a higher number of training rounds results in more energy consumption due to the processing power required for each round.
- **Memory:** The memory consumed by a model is recorded by determining the size of the trained model, as well as the amount of data that was necessary to train the model until convergence. The latter is influenced by the number of training rounds and the amount of local computation.

It is worth noting that each of the aforementioned resources are influenced by the number of training rounds in the FL system. This relationship is intuitively explained by the fact that a higher number of training rounds directly results in a longer device operation time, which automatically means there is more time for communication, more energy consumption and a higher memory usage.

Chapter 7

Results

The previous chapter has described the methodology as applied in this research. The current chapter presents results achieved through said methods. First, the results of parameter tuning is elaborated upon. Then, the results achieved through the different sets are presented per set, highlighting the impact of individual variables.

7.1 Parameter Tuning

Parameter tuning is performed through a grid search for regularisation parameters and an analysis of the impact of the batch size, the outcomes of which are presented in this section.

7.1.1 Regularisation

The values for the regularisation parameter λ of both the L_1 activity regulariser and the L_2 kernel regulariser are fine-tuned through a grid search with four iterations. The values of all analysed L_1 values are $[0, 5*10^{-5}, 0.001, 0.002, 0.0003, 0.0004, 0.0005, 0.0008, 0.001, 0.01, 0.1]$ and those of L_2 are $[0, 0.0001, 0.0003, 0.0005, 0.0007, 0.0008, 0.0009, 0.001, 0.003, 0.01, 0.1]$. The results of the last iteration are presented in Figure 7.1 as F1-score of the corresponding parameter values, as the average of data from all scenarios as presented in Table 4.2. It should be noted that an incorrect loss function was used to get these results, with N as noted in Equation 6.1 fixed at 5 instead of being variable to the number of possible classes. Since the performance results in Figure 7.1 are independent of the loss, the values do represent a realistic F1-score. Based on these results, the L_1 regularisation parameter is set to $5*10^{-5}$ and the L_2 parameter to 0.0008. In order to get an understanding of how the network behaves with the correct loss function, a second grid search was performed, the results of which are presented in Figure 7.2. Minimum and maximum values of all scenarios are shown in Appendix I. It can be observed that the chosen parameter values are not optimal in this setting. These results were however obtained only after the FL experiments were conducted, and therefore parameter values were left unaltered.

A noteworthy observation based on Figure 7.2 is that the evaluation score is greatly influenced by the value of λ for L_1 for the activity regulariser: a value of 0.01 and above

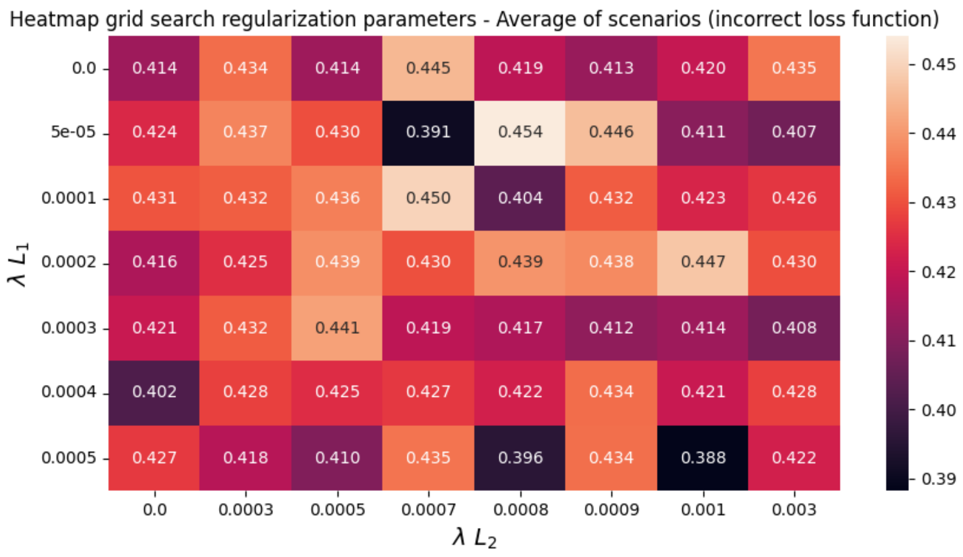


Figure 7.1: Heat map of grid search for regularisation parameter with incorrect loss function, average F1-scores of all scenarios.

results in a low average performance. In contrast, a value below 0.001 does not influence network performance significantly. The regularisation parameter for L_2 for the kernel regulariser contributes less to the performance, as the values in each row of the heat map all lie a maximum of 0.1 away from each other. Additionally, it can be observed that all values of the parameter for L_2 result in a descent score for the L_1 parameter equal to 0, thus allowing for a network without the L_1 activity regulariser.

To evaluate the ability of a network to generalise across different environments, a separate network is optimised with each scenario combination, after which model evaluation is performed on each individual scenario, both with and without network regularisation. The scenario combinations resulting in the highest inference performance for each test set are presented in Appendix J. Figure 7.3 presents a comparison of the F1-scores for models trained with and without L_1 and L_2 regularisation. The number of data points above the diagonal (974) is more or less equal to the total data points below the diagonal (1058).

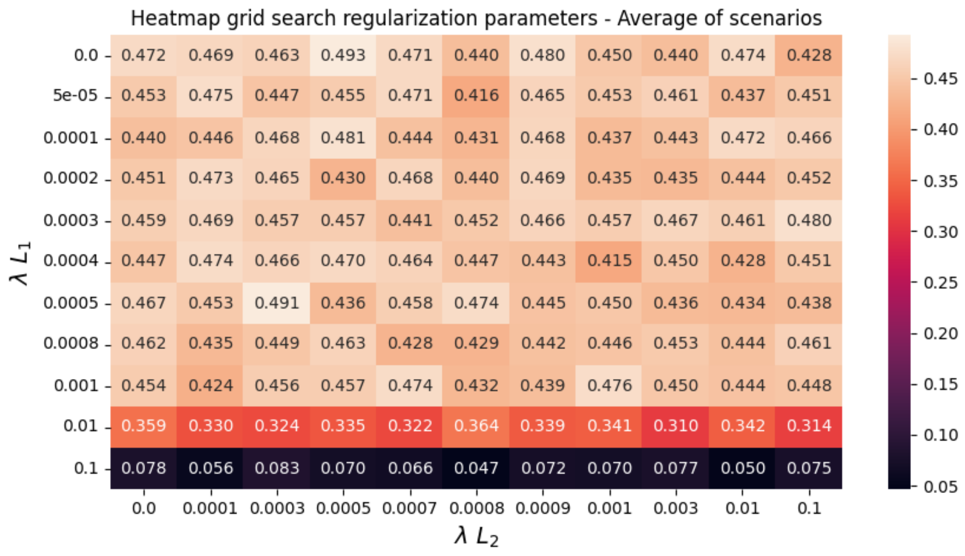


Figure 7.2: Heat map of grid search for regularisation parameter with correct loss function, average F1-scores of all scenarios.

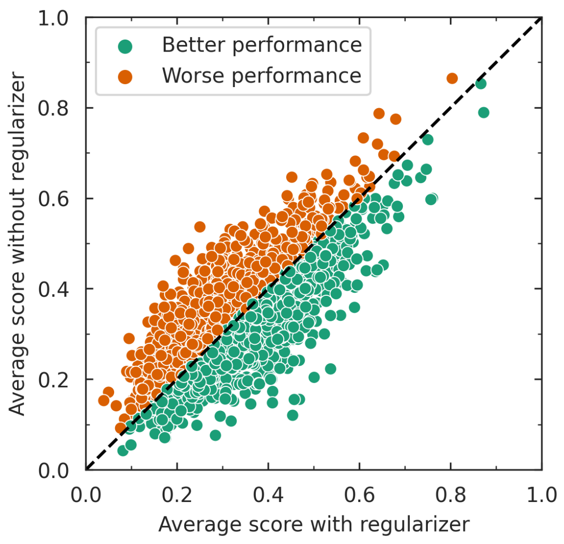


Figure 7.3: Performance comparison optimised network, with and without regularisation.

7.1.2 Batch size

To indicate the impact of the batch size on the network's performance and training convergence, both training accuracy over FL rounds and inference F1-scores are presented for network optimisation using the *simple* set. Figure 7.4 presents training metrics over time and total epochs with a maximum of 1000 epochs. As seen, the wall time is significantly decreased for a higher batch size. No upper bound of the batch size is observed, yet. The total amount of epochs is determined by multiplying the training round number by the number of locally performed epochs in each round. The slopes of the curves in Figure 7.4b are consistent, suggesting that each epoch results in an equivalent amount of learning. In contrast, the amount of learning per second differs between different batch sizes.

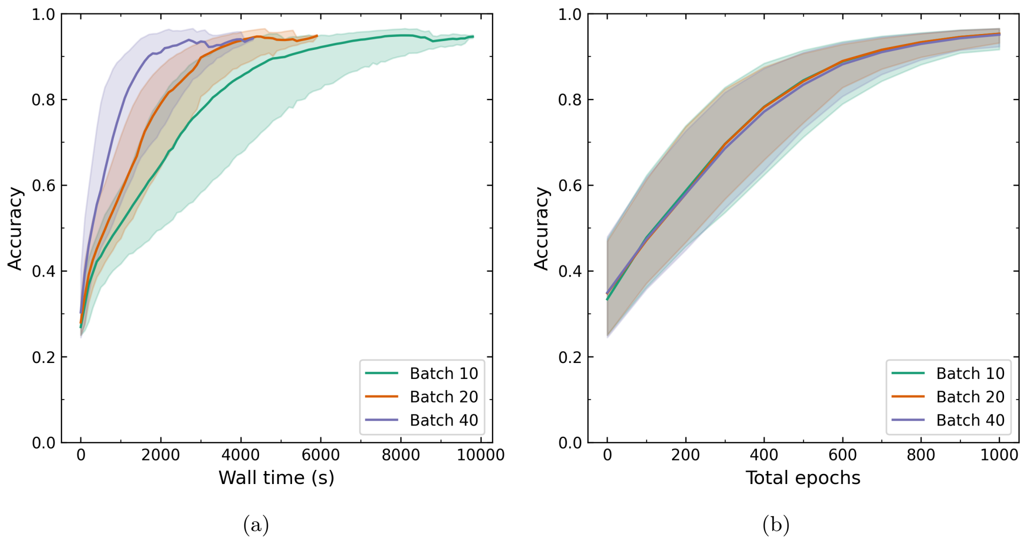
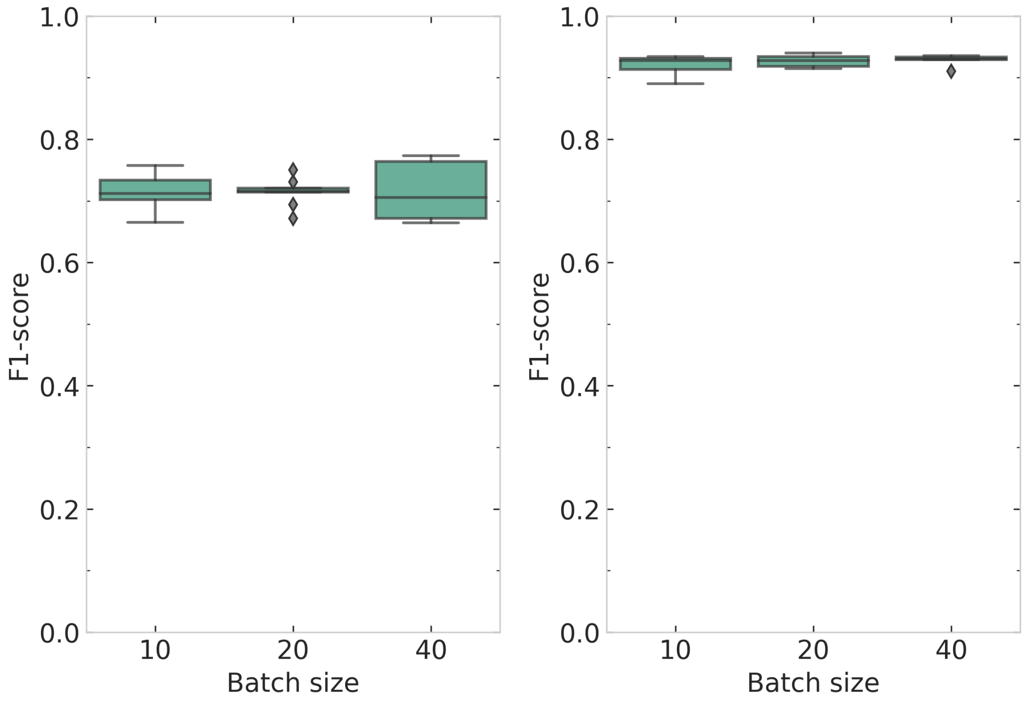


Figure 7.4: Average training accuracy with the transparent band indicating the accuracy range for different number of total epochs, for various batch sizes, with a) categorical training accuracy over time, and b) categorical accuracy over total epochs.

The F1-scores for a total of 500 and 1000 epochs are presented in Figure 7.5. Details on individual measurements can be found in Appendix K.

To indicate the direct impact of different batch sizes on both the F1-score and the convergence time, the difference in F1-score compared to the percentage difference in convergence time is indicated in Figure 7.6 for varying total epochs. The performances with a batch size of 10 and 20 are compared to the performance with a batch size of 40.



(a) (b)

Figure 7.5: Boxplot F1-score comparison over varying local epochs, for different batch sizes and all locations, with a) 500 and b) 1000 total epochs.

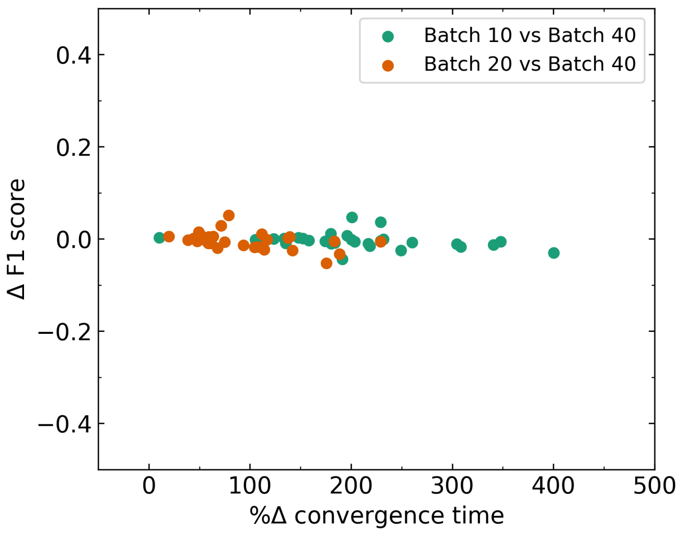


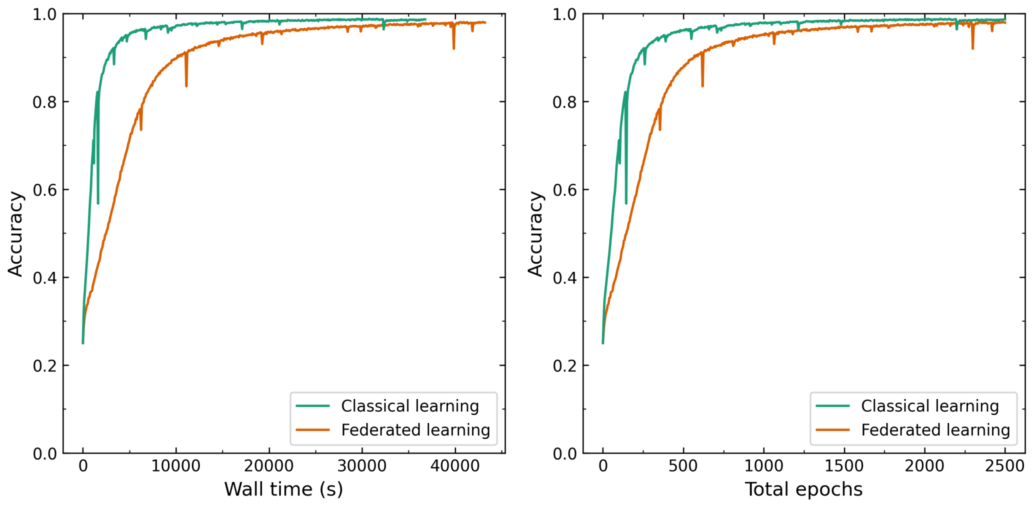
Figure 7.6: Difference in F1-score and convergence time for varying batch sizes compared to a network with batch size 40.

7.2 Effect of Varying Federated Learning Settings

The *complete* set is used to compare the classical ML performance to that of FL, the use of different aggregation algorithms, and the impact that the amount of local computation and communication between clients has on model performances.

7.2.1 Machine learning and federated learning

The accuracy during network optimisation for both the classical ML and FL approach are presented in Figure 7.7. The classical approach is achieved by assigning all available data to a single client, thus being limited to local training only. The batch size for this and all other following training settings is kept constant at a value of 20. It is observed that the classical approach reaches convergence in significantly less time than the federated approach. Table 7.1 shows the evaluation performances of both setups for a varying total number of epochs. The performance of classical ML is consistently higher for different amounts of total performed epochs. The biggest difference is with 250 total epochs and only 1 local epoch per round in the federated setting. The F1-score of FL is more than twice as small as the score achieved through the classical approach.



(a) (b)

Figure 7.7: Training accuracy for federated and classical learning, 500 training rounds and 5 epochs per training round, with a) categorical training accuracy over time, and b) categorical accuracy over total epochs.

Figure 7.8 shows the difference in F1-score and the relative difference in convergence time of the FL system compared to the classical setup. The results of individual training sessions are presented in Appendix L. It is observed that the performance score of FL is always lower than the classical setting without any significant improvement in convergence time.

Table 7.1: Performance comparison federated learning vs. classical machine learning.

Total epochs	**FL rounds**	**Local FL epochs**	**F1-score (classical)**	**F1-score (federated)**
250	250	1	0.929	0.438
500	500	1	0.970	0.755
1250	250	5	0.985	0.934
2500	250	10	0.988	0.954

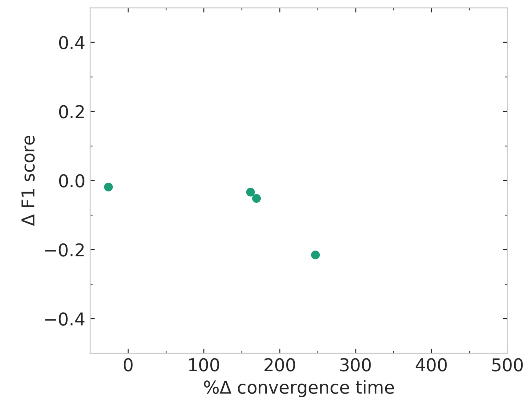


Figure 7.8: Difference in F1-score and number of epochs until convergence for federated learning compared to classical machine learning.

7.2.2 Aggregation algorithm

The training accuracy over time and over the total number of epochs are shown in Figure 7.9. The different curves indicate that the aggregation algorithms do not differ from each other for the total number of training epochs, except for the `MimeLite` algorithm with momentum. The wall time does differ for each algorithm, with `FedProx` with $\mu = 1$ outperforming the rest in convergence speed. Additionally, `MimeLite` without momentum converges in significantly more time than other algorithms, except for `MimeLite` with momentum, which increases rapidly for a small number of epochs, ultimately reaching an accuracy of 0.8 in about the same number of epochs as the other algorithms.

The evaluation scores of the algorithms are presented in Table 7.2. It is interesting to note that the F1-score differs only for a low number of total epochs and only for `MimeLite` with momentum. This is coherent with the initial higher learning rate for a small number of total epochs as previously described.

In Figure 7.10, the differences in F1-score and convergence time compared to the `FedAvg` averaging algorithm are indicated for all other algorithms. Detailed results on convergence time are presented in Appendix M. The `FedProx` algorithm with a sufficiently high μ value performs within 0.1 difference in F1-score and converges up to 18.5% faster than the `FedAvg` method. `MimeLite` with momentum performs better when the model is optimised with only 250 total epochs, even reaching an F1-score of 0.847 without reaching a training accuracy of 0.8. When considering a greater number of total epochs, the convergence rate of `MimeLite` with momentum is observed to be 174.0% slower compared to `FedAvg`. However, `MimeLite` achieves an F1-score that is 0.162 higher than that of `FedAvg`.

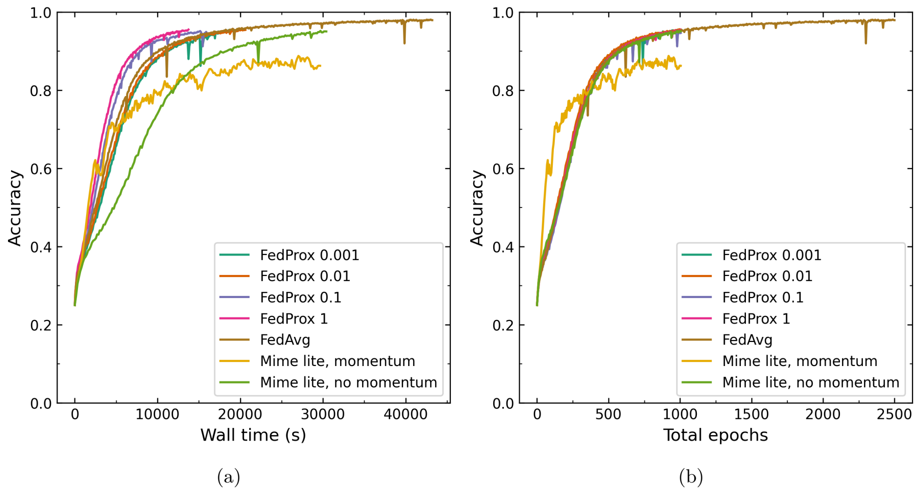


Figure 7.9: Training accuracy for various averaging algorithms, 500 training rounds and 5 epochs per training round, with a) categorical training accuracy over time, and b) categorical accuracy over total epochs.

Table 7.2: Performance comparison different averaging algorithms.

Rounds	**Epochs**	**F1 (FedProx 1)**	**F1 (FedProx 0.1)**	**F1 (FedProx 0.01)**	**F1 (FedProx 0.001)**	**F1 (MimeLite, momentum)**	**F1 (MimeLite, no momentum)**	**F1 (FedAvg)**
250	1	0.417	0.45	0.437	0.474	0.847	0.441	0.438
	250	0.912	0.91	0.897	0.906	0.911	0.901	0.934
	250	0.885	0.889	0.89	0.887	0.867	0.891	0.954
500	1	0.766	0.75	0.742	0.736	0.917	0.76	0.755
	500	0.911	0.9	0.909	0.905	0.896	0.9	0.967

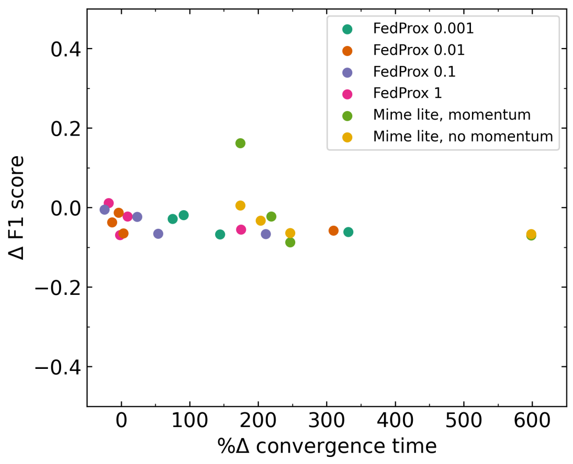
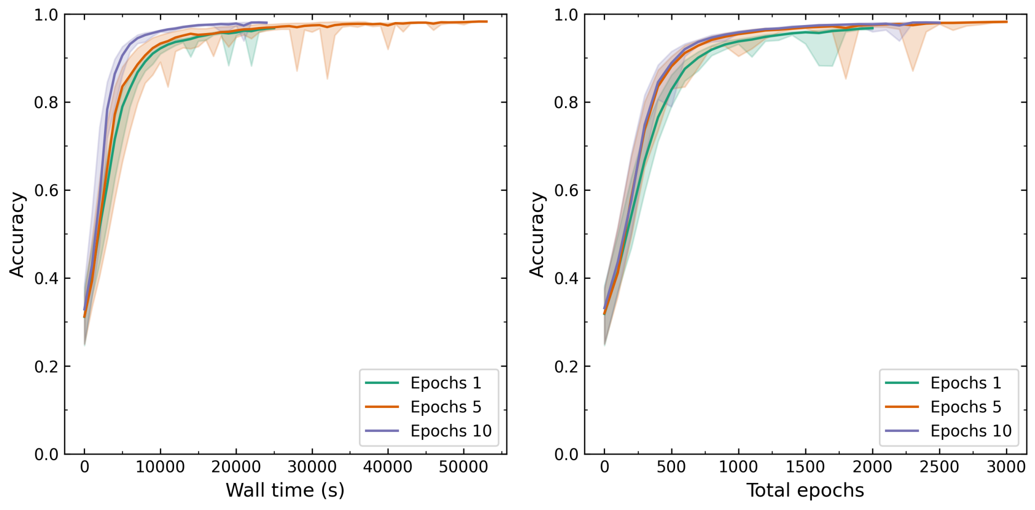


Figure 7.10: Difference in F1-score and convergence time for varying averaging algorithms compared to `FedAvg`.

7.2.3 Computation and communication

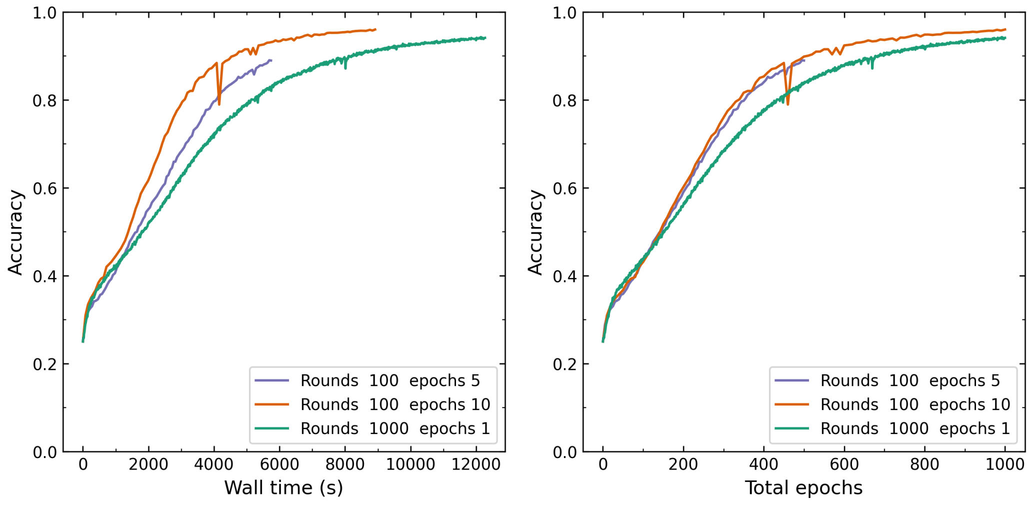
The impact of and the trade-off between computation an communication is depicted in Figures 7.11 and 7.12. The graphs demonstrate the effect of varying the number of epochs per training round, and the impact of maintaining an equal total amount of epochs while varying the amount of local computation, respectively. The number of FL rounds represent the amount of communication between client and server. The predefined maximum budget is set to 3000 epochs. Minor differences in convergence time are perceived for different amounts of locally performed epochs. In general, more local epochs cause a speed up in convergence time, both in wall time and number of total epochs. This can be seen in more detail in Figure 7.12b, in which it is shown that 10 local epochs per FL round allows for faster convergence time than 1 local epoch per round for the same total number of epochs. Note that the scale on the x-axis is different for Figures 7.11 and 7.12. This is done to properly highlight the difference during the optimisation process for a varying amount of local computation.



(a) (b)

Figure 7.11: Average training accuracy with the transparent band indicating the accuracy range for various local epochs and varying total number of training rounds, with a) categorical training accuracy over time, and b) categorical accuracy over total epochs.

Table 7.3 presents the evaluation performances of different amounts of local computation. The improvement in F1-score stagnates for a higher amount of local computation. When comparing scenarios with an equal number of total epochs, it is consistently observed that one epoch per training round outperforms 10 epochs per round, at the expense of increased convergence time.



(a) (b)

Figure 7.12: Training accuracy for a total of 500 and 1000 epochs, and varying number of local epochs, with a) categorical training accuracy over time, and b) categorical accuracy over total epochs.

Table 7.3: Performance comparison for a varying amount of local computation.

Total epochs	**Epochs**	**Rounds**	**F1-score**
500	1	500	0.755
	5	100	0.760
	10	50	0.733
1000	1	1000	0.935
	10	100	0.888
2500	5	500	0.967
	10	250	0.954

7.3 Effect of Adding New Locations

This section presents results of network optimisation in a changing environment, with a newly introduced location at which activities are executed using the *location* set. The optimised models are subsequently trained with unseen data under variable parameters, allowing for analysis of a model's capability to adapt to new environments. Training and evaluation metrics are presented for each configuration.

7.3.1 Change in location

A change in location is simulated by introducing a retained location at which an activity was performed. Below, first the training and evaluation metrics are presented of the optimisation process on two locations. After, the impact of the unseen data is shown. The score of the best performing model is made bold. This is consistently the benchmark score.

Optimisation with two locations

Using the *location* set, classification models are optimised based on two out of the three available locations where activities were performed, for varying number of training rounds, local epochs and batch size 20. The model performance for each model that is trained without a certain location is shown in Table 7.4 for varying number of total epochs. The benchmark scores are based on the *complete* set, without removal of locations.

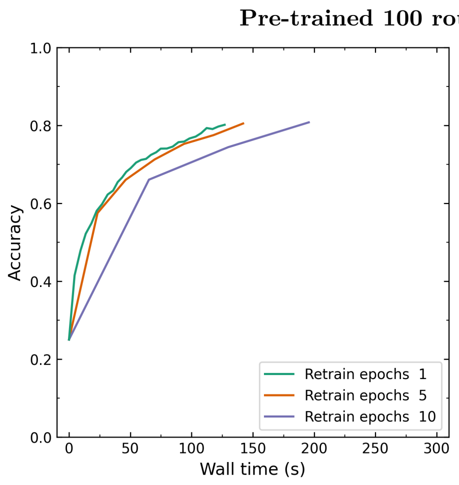
Table 7.4: Performance comparison with removed location.

Total epochs	**Epochs**	**Rounds**	**F1-score (without $\mathbf{L_{TV}}$)**	**F1-score (without $\mathbf{L_{table}}$)**	**F1-score (without $\mathbf{L_{kitch}}$)**	**Benchmark**
500	1	500	0.563	0.589	0.594	**0.755**
	5	100	0.539	0.593	0.599	**0.760**
	10	50	0.550	0.595	0.602	**0.733**
1000	1	1000	0.715	0.714	0.718	**0.935**
	10	100	0.718	0.718	0.718	**0.888**
2500	5	500	0.713	0.714	0.718	**0.967**
	10	250	0.744	0.736	0.715	**0.954**

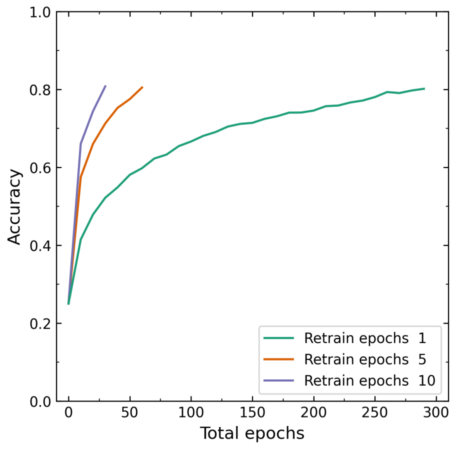
Unseen location

The training accuracy for network optimisation on an unseen location is presented in Figure 7.13 for location L_{TV}. The network was previously trained on the other two locations for either 100 or 500 training rounds and 10 local epochs per round. Similar behaviour was observed for the other two unseen locations. Training on unseen data is terminated after achieving a training accuracy of 0.8 or higher. The curve indicates that again, more local computation results in less necessary epochs until network convergence. This is seen by comparing *Retrain epochs 10* to *Retrain epochs 5* and *Retrain epochs 1*. However, the wall time until convergence is higher for more local epochs when only a small amount of total epochs is necessary. Figure 7.14 depicts training accuracy for the same training configurations as Figure 7.13, with the only difference being the number of local epochs per training rounds. This number has decreased from 10 to 1 epoch per round. More total epochs until convergence are observed for each amount of locally performed epochs during re-training. Interestingly, the total training time is similar for each *Retrain epochs 10*, *Retrain epochs 5*, and *Retrain epochs 1*.

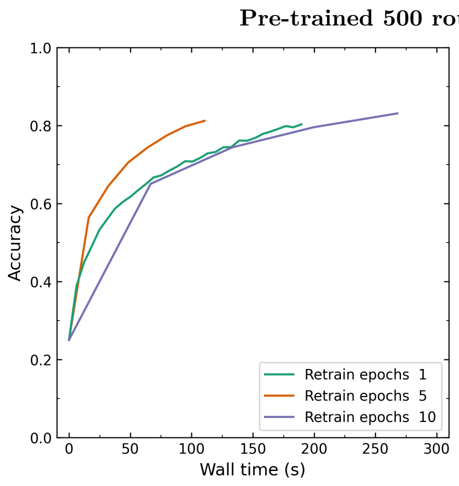
The amount of epochs for which the pre-trained model was optimised is recorded and compared to both the F1-score of the model trained on the new location, and the optimisation time until convergence is reached, i.e. an accuracy of 0.8. The relation between the variables is depicted in Figure 7.15. F1-scores and convergence time of individual runs can be found in Appendix N. From the figure, it is observed that when models are pre-trained with relatively few training rounds, evaluation performance is lower up to a certain number of pre-trained epochs, and it takes a longer amount of time until convergence is reached when new data are added to the system.



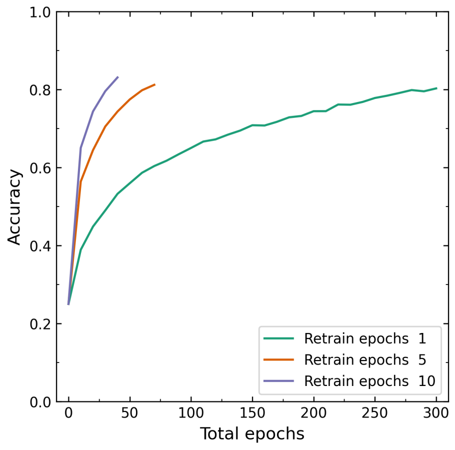
(a)



(b)

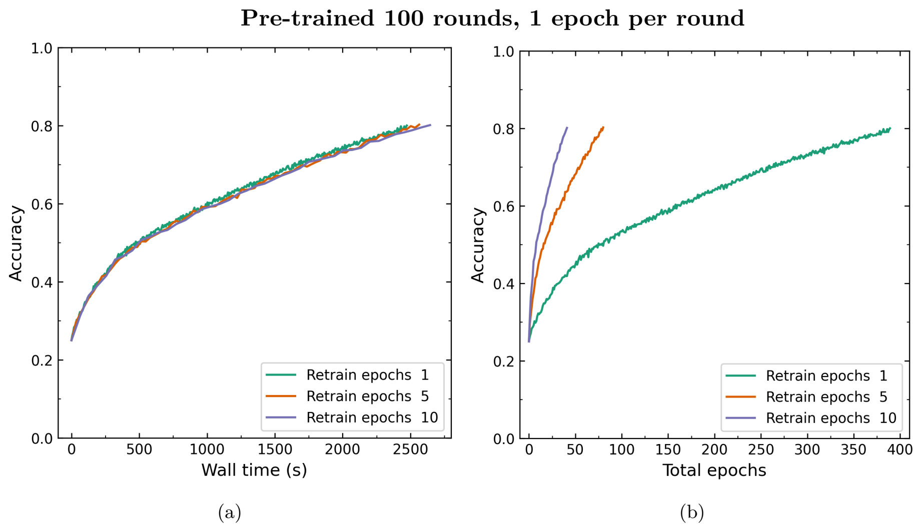


(c)



(d)

Figure 7.13: Training accuracy for continuing training a network on unseen location L_{TV} with varying number of epochs per round, pre-trained with 100 or 500 rounds and 10 epochs per round. In a) categorical training accuracy over time, and b) categorical accuracy over total epochs, both pre-trained with 100 training rounds. In c) categorical training accuracy over time, and d) categorical accuracy over total epochs, both pre-trained with 500 training rounds.



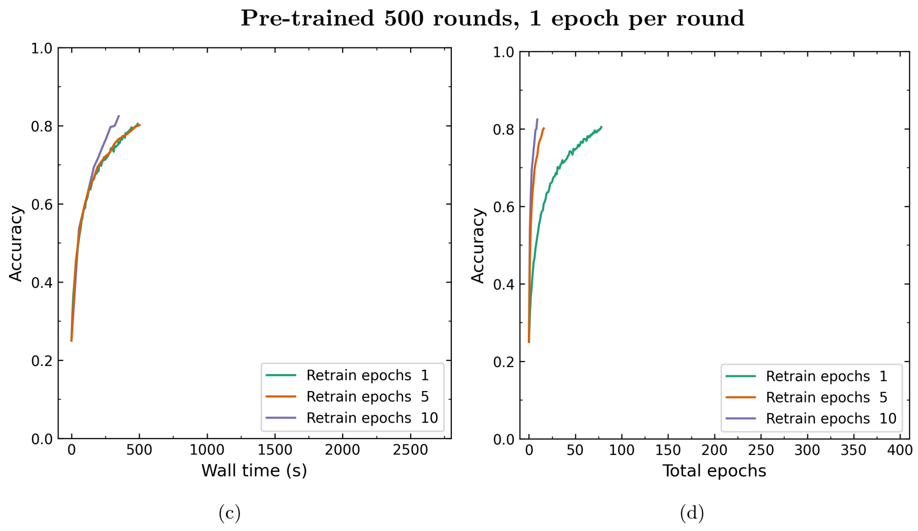
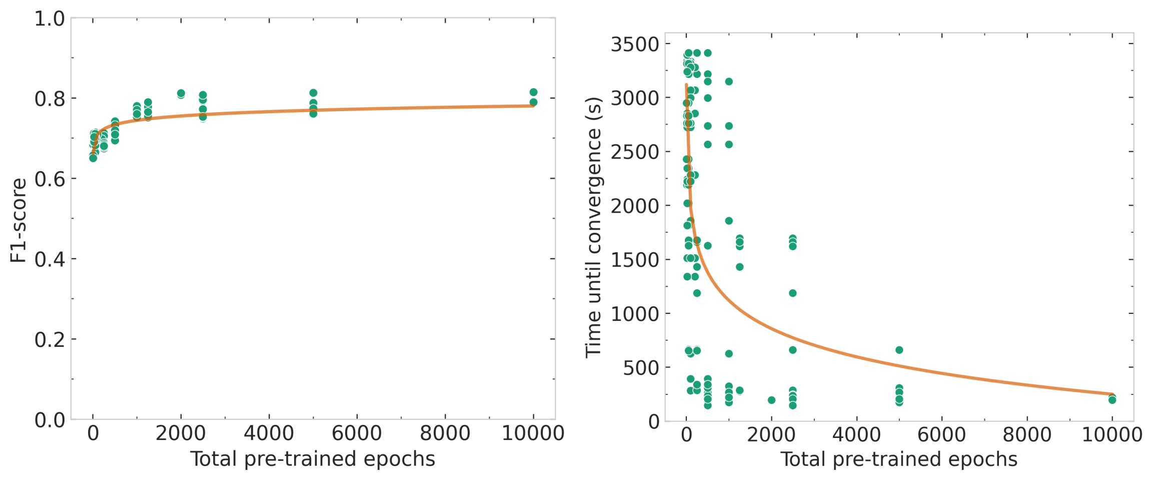


Figure 7.14: Training accuracy for continuing training a network on unseen location L_{TV} with varying number of epochs per round, pre-trained with 100 or 500 rounds and 1 epoch per round. In a) categorical training accuracy over time, and b) categorical accuracy over total epochs, both pre-trained with 100 training rounds. In c) categorical training accuracy over time, and d) categorical accuracy over total epochs, both pre-trained with 500 training rounds.



(a) (b)

Figure 7.15: Impact of amount of pre-training done on model for new location with a curve fit indicating the trend, with a) the impact on F1-score, b) the impact on the amount of training time until convergence is reached.

7.3.2 Limited data availability

A fraction of the data is taken with which network optimisation is continued on a pre-existing model until a training accuracy of 0.8 is reached. The data fractions are $[0.1, 0.2, 0.5, 0.75, 1]$. As in previous configurations, re-training is performed with different amount of local computation. The impact of the available data on the F1-score and the total amount of epochs necessary until convergence is reached is depicted in Figure 7.16. More details on the impact on F1-score and convergence are presented in Appendix P. When only 10% of the data are available, a lower F1-score is observed compared to considering 50% or more of the data. Notably, using only 50% of the available data achieves equivalent performance to using all available data. The same behaviour is not perceived in the number of epochs required to reach convergence. Leveraging all available data results in a faster convergence compared to using only a fraction of the data. Figure 7.17 illustrates the trade-off between performance, convergence speed, and the quantity of data considered during optimisation, also indicating this behaviour. This figure shows the relation between the difference in F1-score and convergence time for each data fraction compared to training with all available data. For example, the entry “0.1 vs 1” shows, for each model re-trained with 10% of the available data, the difference in F1-score between it and a model re-trained with 100% of the available data, plotted against the percentage difference in convergence time between both models.

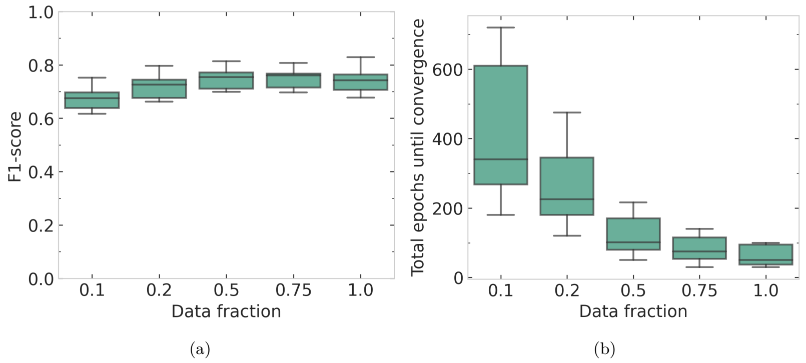


Figure 7.16: Impact of amount of data available during optimisation on pre-existing models (*location* set), with a) the impact on F1-score, b) the impact on the amount of total training epochs until convergence is reached.

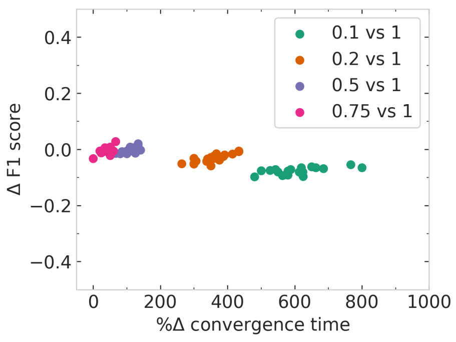


Figure 7.17: Difference in F1-score and number of epochs until convergence for varying data fractions compared to a network trained with all data (*location* set).

7.4 Effect of Adding New Clients

This section presents network results with the absence as well as introduction of one of the clients n_i using the *client* set. F1-scores and convergence time of individual runs can be found in Appendix O.

7.4.1 Change in client participation

A change in participating clients can occur when new devices enter a system. This situation is simulated through the introduction of a new client, which transmits new signals to all pre-existing clients. This also goes for the opposite direction: the new client starts receiving data from pre-existing clients. This introduces a new communication channel into the system. The following results present the metrics of training on three clients, thus excluding a fourth client, after which the results of different configurations of the new client are shown. The latter is done via the three different schemes as described in Chapter 6.

Optimisation with three clients

Based on the *client* set, network optimisation is performed with the use of three out of four nodes at which CSI data are measured. The results are presented in Table 7.5 for each excluded client. The score of the best performing model is made bold. Again, the benchmark scores are based on the *complete* set. It is observed that the performance of the models trained without client n_{AP} is consistently higher than the benchmark score. It should be noted that the test set used to determine the F1-score also did not contain data from n_{AP}. It can be stated that the data measured by this client are relatively noisy and therefore increases data complexity.

Table 7.5: Performance comparison with removed client.

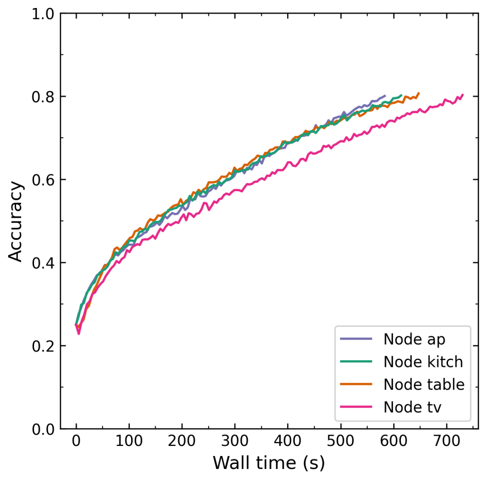
Total epochs	**Epochs**	**Rounds**	**F1-score (no $\mathbf{n_{AP}}$)**	**F1-score (no $\mathbf{n_{TV}}$)**	**F1-score (no $\mathbf{n_{kitch}}$)**	**F1-score (no $\mathbf{n_{table}}$)**	**Benchmark**
500	1	500	**0.863**	0.787	0.808	0.788	0.755
	5	100	**0.863**	0.824	0.828	0.798	0.760
	10	50	**0.888**	0.838	0.784	0.793	0.733
1000	1	1000	**0.968**	0.954	0.948	0.940	0.935
	10	100	**0.976**	0.966	0.947	0.948	0.888
2500	5	500	**0.994**	0.992	0.990	0.989	0.967
	10	250	**0.994**	0.992	0.990	0.987	0.954

New client - receiver only

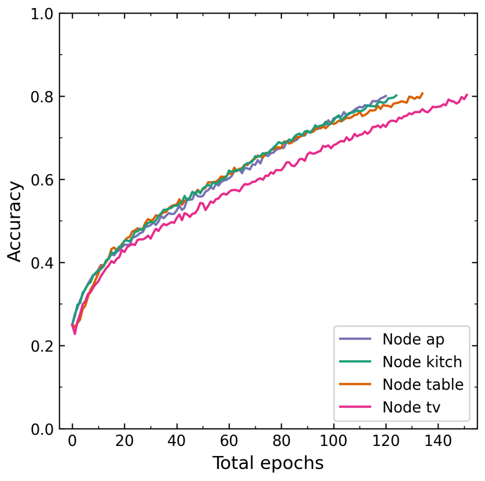
When a new client can act as receiver only, it gathers data from the transmitting nodes around it. These data can be used to train a model locally, starting with a model that is adapted to the environment the node enters. Figure 7.18 shows the local training time and training rounds for each node for models pre-trained on respectively 100, 500 and 1000 training rounds. The training accuracies during the optimisation process in all figures indicate that the newly added n_{TV} is always slower to reach convergence than other nodes, regardless of the amount of pre-training performed by the nodes that were already present. Most notably is the negative learning rate of n_{TV} with 1000 pre-trained rounds. Additionally, where the removal of n_{AP} from pre-training improved the model performance, no optimisation struggle is perceived for this node when added as receiver. In contrast, n_{TV} converges the slowest when added as receiver, without indicating model improvement during the pre-training phase when this node is omitted.

Figure 7.19 shows the F1 evaluation score for different nodes and varying number of pre-trained epochs. The figure indicates that the amount of pre-training affects model performance, with more pre-training resulting in a higher F1-score. Up to 2000 pre-trained epochs was applied, but the performance of various nodes is still improving.

Pre-trained 100 rounds

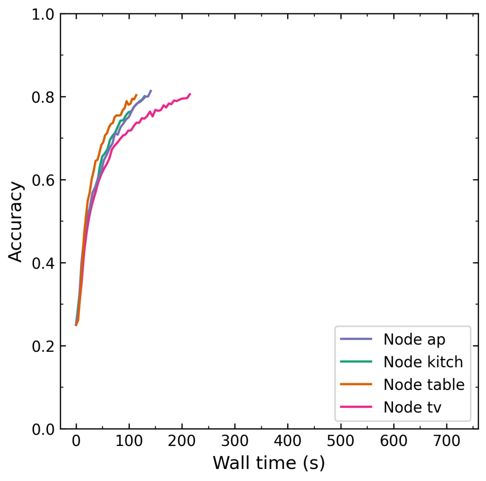


(a)

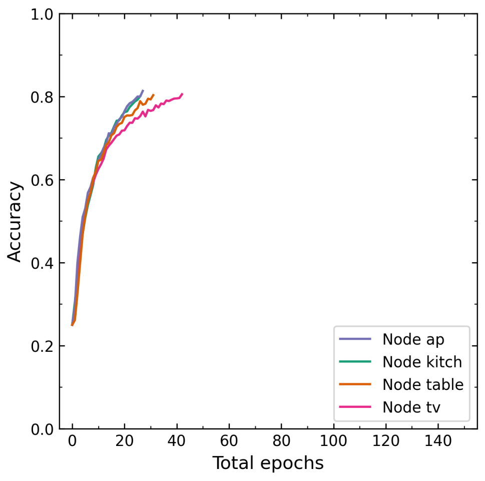


(b)

Pre-trained 500 rounds

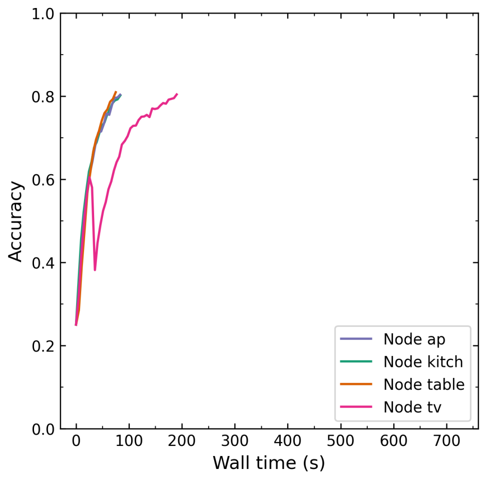


(c)

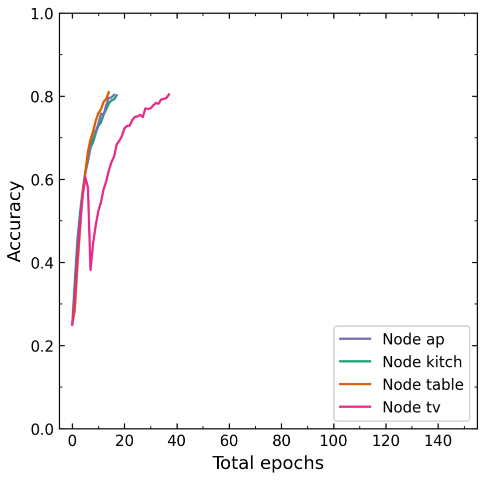


(d)

Pre-trained 1000 rounds



(e)



(f)

Figure 7.18: Training accuracy for continuing training a network locally on unseen nodes (receiver only) with data from three pre-existing nodes, pre-trained with 100, 500 and 1000 rounds, and 1 epoch per round.

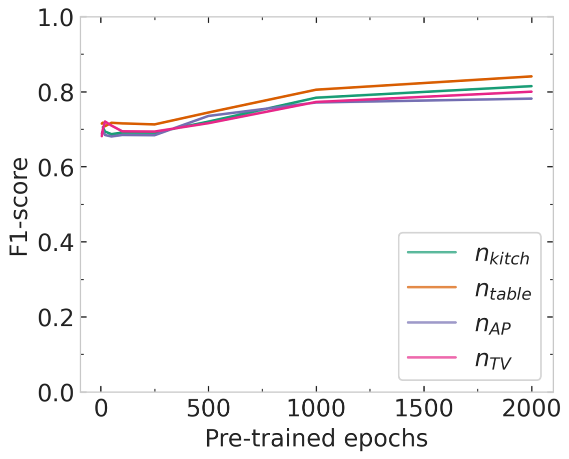


Figure 7.19: Comparison F1-scores for different receiver nodes added to the network.

New client - transmitter only

A node that is only able to transmit CSI data can not participate in an FL configuration, as there is no locally available data the node can contribute with. Instead, in the following results the new client solely introduces new data into the system by transmitting to pre-existing clients. The graphs in Figure 7.20 below depict the training time and total number of training rounds of the FL system for differently pre-trained models. The node as indicated in the legend represents the excluded node. From Figure 7.20 it can be observed that the addition of n_{AP} as transmitter results in a longer converge time of the network compared to adding the other nodes.

Training continuation was also performed with an increased amount of pre-training, which shows similar results to those presented earlier for the *location* set in a varying amount of local computation. Again, a higher number of pre-trained epochs results in less training time necessary until convergence is reached. Furthermore, the performance of the model increases, as indicated in Table 7.6. The F1-scores made bold are those scores that are the highest among all nodes with the same pre-trained epochs and rounds.

Table 7.6: Performance comparison for different transmitter nodes added to the network.

Total pre-trained epochs	**Pre-trained epochs**	**Pre-trained rounds**	**F1-score (added $\mathbf{n_{AP}}$)**	**F1-score (added $\mathbf{n_{TV}}$)**	**F1-score (added $\mathbf{n_{kitch}}$)**	**F1-score (added $\mathbf{n_{table}}$)**
500	1	500	0.699	0.710	**0.723**	0.706
	5	100	0.704	**0.748**	0.726	0.710
	10	50	0.708	**0.748**	0.712	0.704
1000	1	1000	0.735	**0.777**	0.756	0.764
	10	100	0.736	**0.802**	0.788	0.766
2500	5	500	0.770	**0.812**	0.793	0.793
	10	250	0.790	0.825	0.806	**0.835**

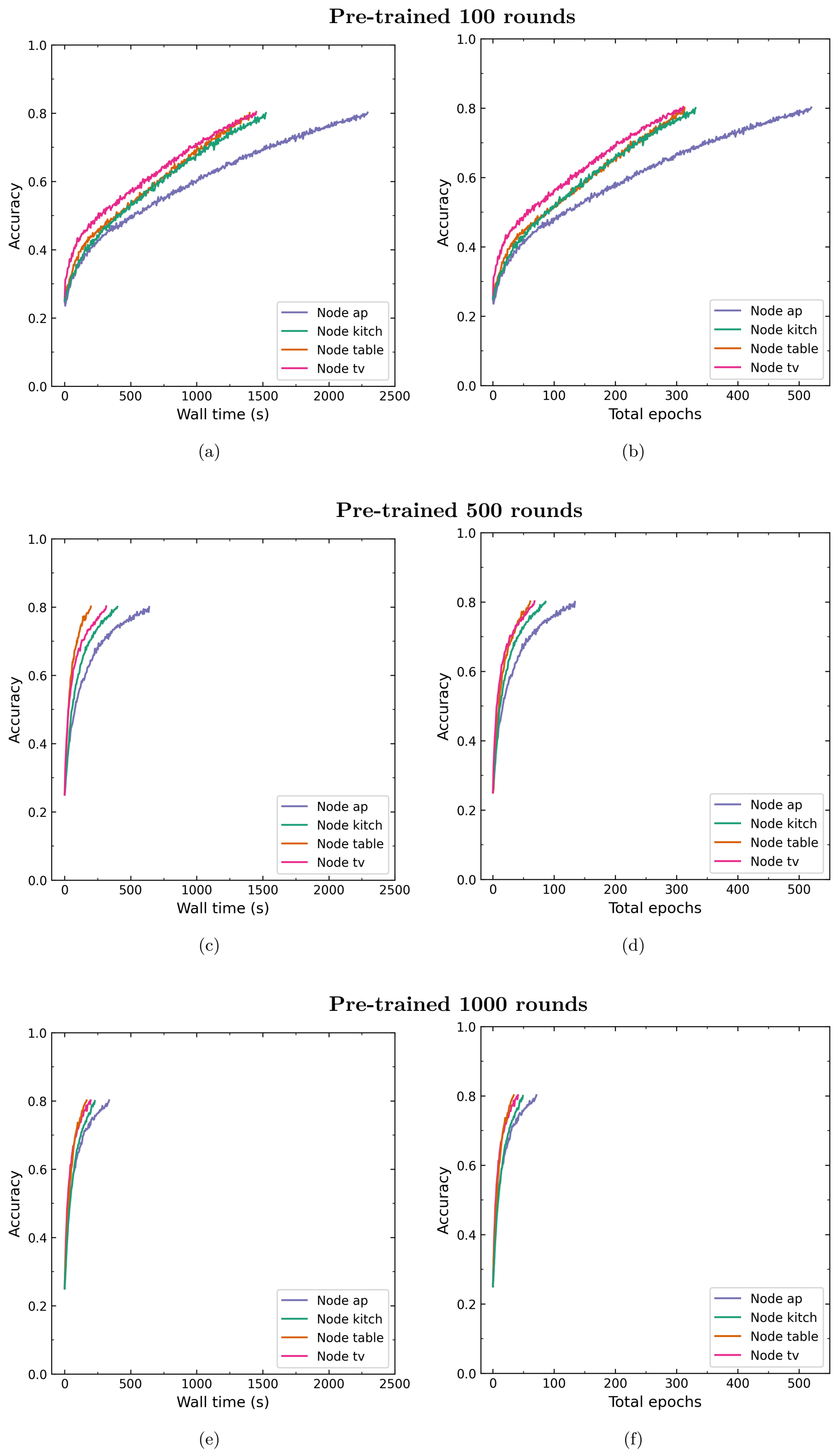


Figure 7.20: Training accuracy for continuing training a network in an FL setting on pre-existing nodes with data from an unseen node (transmitter only), pre-trained with 100, 500 and 1000 rounds, and 1 epoch per round.

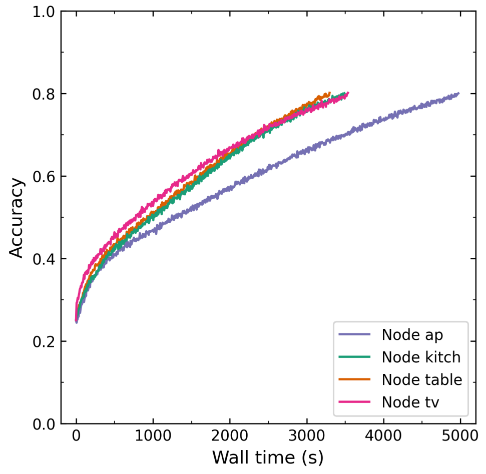
New client - complete network

Lastly, the newly added node was allowed to both sent and receive data, enabling it to contribute to the FL system. The data both sent and received by the new node are now used to continue training on a pre-trained network. Figure 7.21 presents the optimisation accuracy for both time and total number of performed training rounds for each newly added node. Table 7.7 presents evaluation scores for a selection of pre-trained number of epochs. Similar behaviour and differences between nodes are observed as described before. Again, training continuation was also performed for a varying amount of local computation. These results are not shown here, as the same behaviour was exhibited as previously presented results for variation in local epochs. The best performing model for each row is made bold. The node n_{TV} performs best in most cases.

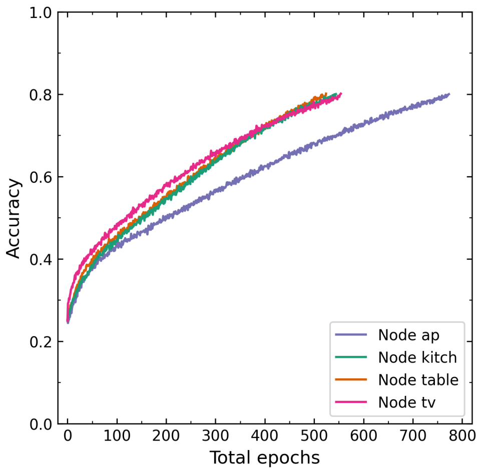
Table 7.7: Performance comparison for different nodes added to the network, both acting as receiver and transmitter.

Total pre-trained epochs	**Pre-trained epochs**	**Pre-trained rounds**	**F1-score (added $\mathbf{n_{AP}}$)**	**F1-score (added $\mathbf{n_{TV}}$)**	**F1-score (added $\mathbf{n_{kitch}}$)**	**F1-score (added $\mathbf{n_{table}}$)**
500	1	500	0.702	0.719	**0.720**	0.716
500	5	100	0.728	**0.736**	0.723	0.724
500	10	50	0.715	**0.737**	0.727	0.726
1000	1	1000	0.734	0.754	0.758	**0.759**
1000	10	100	0.744	**0.783**	0.754	0.763
2500	5	500	0.753	**0.790**	0.783	0.781
2500	10	250	0.779	**0.789**	0.783	0.785

Pre-trained 100 rounds

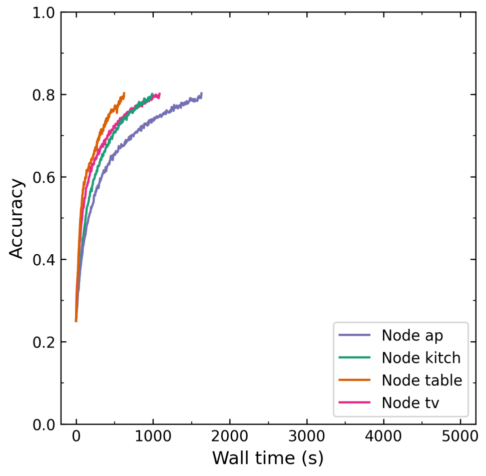


(a)

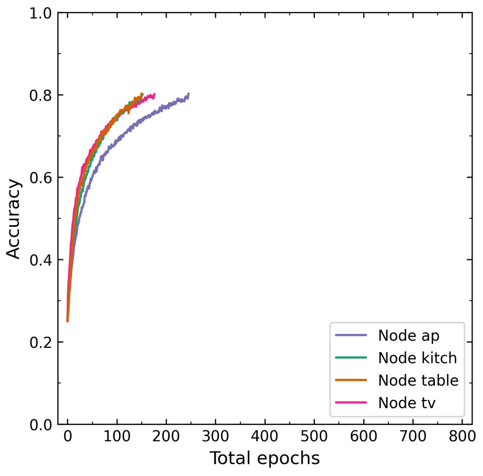


(b)

Pre-trained 500 rounds

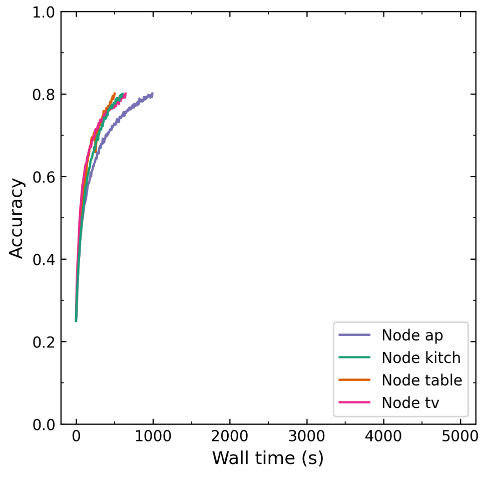


(c)

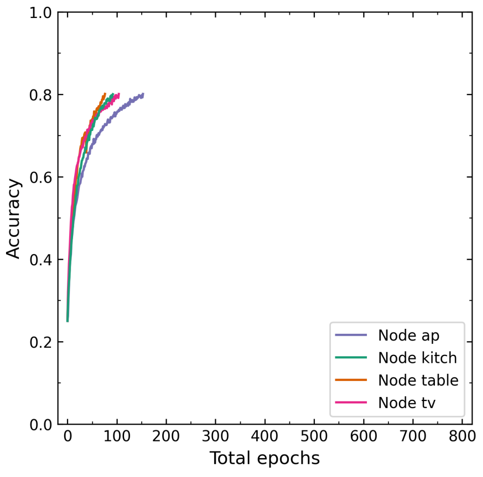


(d)

Pre-trained 1000 rounds



(e)

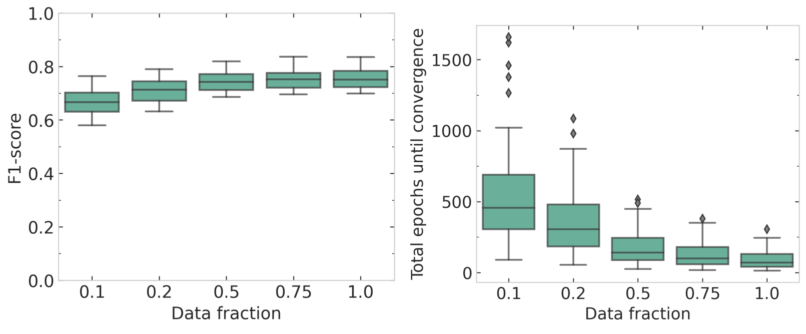


(f)

Figure 7.21: Training accuracy for continuing training a network in an FL setting on pre-existing nodes with data from an unseen node (complete network), pre-trained with 100, 500 and 1000 rounds, and 1 epoch per round.

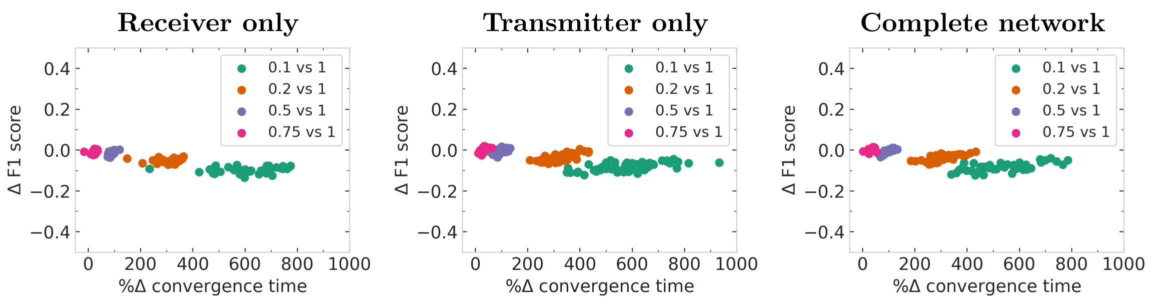
7.4.2 Limited data availability

A fraction of the data is taken with which network optimisation is continued on a pre-existing model until a training accuracy of 0.8 is reached. The data fractions are $[0.1, 0.2, 0.5, 0.75, 1]$. As in previous configurations, re-training is performed with different amount of local computation. The impact of the available data on the F1-score and the total amount of epochs necessary until convergence is reached is depicted in Figure 7.22. More details on the impact on F1-score and convergence are presented in Appendix P. The results indicate that again, the limited availability of data negatively impacts the F1-score and the convergence speed. Figure 7.23 indicates the relation between a difference in F1-score and difference in convergence time for the data fractions, as compared to continuation of training a network with all available data. The trade-off can be seen that was also mentioned earlier between performance, convergence time, and data fraction for the three different setups with newly added clients. Figure 7.23 indicates the relation between a difference in F1-score and difference in convergence time for the data fractions, as compared to continuation of training a network with all available data. A lower data fraction results in relatively longer convergence time and lower F1-score for each scheme.



(a) (b)

Figure 7.22: Impact of amount of data available during optimisation on pre-existing models (*client* set), with a) the impact on F1-score, b) the impact on the amount of total training epochs until convergence is reached.



(a) (b) (c)

Figure 7.23: Difference in F1-score and number of epochs until convergence for varying data fractions compared to a network trained with all data (*client* set).

Chapter 8

Discussion

In the previous chapter, experiment results have been presented for each data set. The following chapter aims to discuss these results and provides a detailed explanation of the observations. Next, limitations of the research as well as implementation flaws are highlighted.

8.1 Parameter Tuning

This section discusses the results of the regularisation parameter tuning and the effect of varying the batch size.

8.1.1 Regularisation

Based on the regularisation results, it is implied that fine-tuning the activity regulariser in the network is more important than doing so for the kernel regulariser. Additionally, a low value for the regularisation parameter in the L_1 activity regulariser is desired. As noted earlier, the F1-scores for different regularisation values using the incorrect loss function resulted in slightly different optimal regularisation values. This is possibly caused by data randomness. It is argued that this has not impacted results greatly, as all performance values for L_1 parameter equal to or lower than 0.001 lie within 0.10 of each other, and relative comparisons can still be made.

The comparison presented in Figure 7.3 indicates that including L_1 and L_2 regularisation does not lead to obvious improvement compared to the absence of regularisation. It should be noted that dropout was implemented in either case. These results are based on data from the pilot experiment, and it is expected that because of the increase in complexity and amount of the data gathered in the eHealth House, regularisation is more important.

As noted in Section 6.1, L_1 activity regulariser could not be implemented in the FL model. It is expected that the results were not affected significantly by this, as it is equivalent to regularisation with parameter λ equal to 0, which always results in a descent F1-score regardless of the value of the L_2 parameter.

8.1.2 Batch size

The batch size in FL settings influences the total time needed to complete a certain amount of training rounds. The decrease in wall time for smaller batch sizes is caused by a decrease in necessary computations per epoch, as more data are processed per forward and backward pass during model optimisation. Since in Figure 7.4a no upper bound in convergence time is observed, yet, a higher batch size could be considered as long as the device's available local memory allows to do so.

Based on the difference between Figures 7.4a and 7.4b, the batch size only influences the computing time per round, while the amount of learning per epoch remains equal. This is possibly explained by the fact that the amount of data seen per round is the same for different batch sizes, as clients iterate through the entire local dataset for each epoch before communicating the updated model. The batch size only influences the number of samples used per forward and backward pass. This observation contradicts what is observed in literature, which states that smaller batch sizes lead to greater gradient variance due to inconsistent updates across clients, and thus more total epochs are necessary until model convergence is reached [90]. However, data across devices in an FL system are usually non-IID. In this research, data are in fact identically distributed since each device measures the same activities. Therefore, it is suspected that smaller batch sizes do not lead to gradient variance in FL for devices in the same room. The same reasoning can be applied to the lack of difference in F1-score for different batch sizes. Literature shows that higher batch sizes generally lead to lower model performance, yet the performance in this research for different batch sizes is equal, as indicated in Figures 7.5 and 7.6. It is argued that this is caused by the fact that different clients hold similar data, thus counteracting the decrease in learning performance due to a higher batch size, by combining multiple models from different clients.

8.2 Machine Learning and Federated Learning

8.2.1 Performance comparison

The results in Figure 7.7 show the difference between classical ML and FL during network optimisation. It should be noted that the federated process did in fact not reach convergence within 250 training rounds, explaining the relatively low performance.

The relatively high performance achieved with the classical approach is likely caused by the fact that the data are quite similarly distributed across clients. Would the data be more non-IID, FL would be more beneficial. Next to that, additional advantages come with the use of FL, as discussed below.

8.2.2 Communication and privacy

From the performance comparison above it seems there is no good reason to make use of FL in HAR settings. However, as highlighted in Chapter 2, FL enables clients to keep their measured data locally stored, without having to share data to a central server in order to perform activity classification. By sending and receiving updates from the server, knowledge is still shared between clients. In contrast, a classical, non-distributed ML approach would need clients to share all of the measured data. This not only floods communication channels, but consumes resources for both clients and the central server. In addition, future implementations of indoor CSI-based HAR might include inter-household model sharing to improve classification ability to generalise. A classical approach raises privacy concerns when sharing CSI data among unfamiliar entities, as those with malicious intentions can exploit such data to continuously monitor people inside their homes without their consent.

8.3 Federated Learning Settings

In the previous section, differences between local machine learning and federated learning have been highlighted. Now, specific FL settings regarding aggregation algorithms and the amount of local device computation are elaborated upon, and differences are highlighted.

8.3.1 Aggregation algorithms

The similarities between different aggregation algorithms in Figure 7.9 are dedicated to the fact that data are identically distributed across clients and thus there is not much benefit from algorithms dealing with data heterogeneity. Would the data be non-IID across clients, `FedProx` would likely outperform `FedAvg` [38]. It is hypothesised that the minor difference in wall time for `FedProx` with $\mu = 1$ is caused by the fact that penalising client updates results in more time-efficient aggregation. Details are unknown and are likely caused by TensorFlow optimisations. It should be noted that differences may have also been caused by the extent to which the external server utilised for the computations was occupied. Similar behaviour has not been observed in literature, partly because most studies only compare the total number of epochs with each other. The fact that `MimeLite` without momentum converges slower than most other algorithms is attributed to the fact that `MimeLite` without momentum simply reduces to `FedAvg` with more overhead due to the optimiser state being shared with the clients.

The addition of momentum allows for accelerated model updates when the loss is still far from being minimised, but causes the network to slowly converge when the weights approach their target value, possibly due to an insufficiently small learning rate or due to overshooting the optimal solution. This would explain the slow increase in accuracy for `MimeLite` with momentum after a quick start It is suggested to explore the effect of different momentum parameters, as well as including an adaptive learning rate in order to speed up convergence in later stages.

Besides the performance differences between `MimeLite` with momentum and other algorithms, the model size differed, as well: the model size when optimising with momentum, is twice as big compared to doing so without: 13.9 MB compared to 6.9 MB. Accumulating the moving average of preceding model updates requires significantly more memory resources. A trade-off thus exists between convergence time, performance, and required memory. All of the aggregation algorithms eventually result in a properly performing model, and therefore the final choice is based on time constraints and performance requirements depending on specific application demands.

8.3.2 Computation and communication

The differences in convergence time for different amount of local computation is reasoned to be caused by the decrease in aggregation overhead resulting from a reduced number of updates by the central server. Based on this, it is implied that in indoor HAR using FL, an increase in local computation allows for faster convergence due to a decrease in communication. It is important to note that an increased amount of local computation requires more local resources of the participating clients. Therefore, the possible amount of local computation as well as communication between nodes and the central server are device dependent and are influenced by the cost of computation and communication. Currently, communication delay is not implemented in TensorFlow Federated, meaning that this research has not recorded overhead in communication between client and server. In real life implementations, communication costs would be present, increasing the importance of local computation.

Based on the stagnating F1-scores after a certain amount of local computation, it is implied that an excessive amount of local computation per training round is possible and leads to the model overfitting to the data measured by individual clients.

8.4 Varying Activity Location

This section discusses the results achieved through the *location* set and how a varying activity location is dealt with.

8.4.1 Performance with two locations

Before training an existing model on data from an unseen location, the models are trained on data from the two other locations. The models perform similarly for each omitted location. Additionally, all models perform worse than the benchmark mode, which is most probably caused by the reduction in variety in training data.

8.4.2 New locations

For each omitted location, the network is again optimised with data from activities executed in that location. Due to the fact that an increase in local computation also results in an increase the training time per training round, less local computation can actually result in faster convergence. This appears to only be true when only a small amount of training is necessary, since from Figure 7.14 it is apparent that a higher amount of local computation again leads to faster convergence. When less pre-training is performed and thus more re-training is necessary, the amount of local computation matters less, implying that local computation does not influence training continuation on existing models by a great amount.

In real-time applications it is desired to adapt quickly to new environments. Since networks pre-trained with a higher amount of training rounds converge quicker for new data, an FL framework in a changing environment benefits from pre-training a model sufficiently when time and resources allow to do so. A trade-off exists between the convergence time and the number of pre-trained epochs. Again, the specific requirements for different applications influence how much pre- and re-training is possible and necessary.

8.4.3 Performance with limited data

Finally, the availability of training data for the continuation of the training process based on pre-trained models significantly influences both performance and convergence time. When devices have the capacity to store and utilise a relatively large amount of data during the optimisation process, it is recommended to train on new locations with as much data as possible. However, when memory and computational resources are limited, a decision needs to be made on the required performance and convergence speed. It should be noted that the performance reached through leveraging a fraction of 10% of the data might still be acceptable in certain applications.

8.5 Varying Client Participation

This section discusses results achieved through the *client* set, which was used to add new clients to existing networks.

8.5.1 Performance with three clients

Most notably from the results is the noise introduced by client n_{AP}. The noise observed by this client is most probably caused by the placement of the node, at a corner and elevated from the ground. Data considered with this node might therefore by scattered in a greater manner than it does so for other nodes, resulting in noisy CSI data. It is advised to perform more detailed signal propagation analysis to confirm this hypothesis.

8.5.2 New client - receiver

A newly introduced client acting as receiver can perform local computation only on CSI data transmitted by the other clients. The fact that adding n_{TV} added as new node is slowest to reach convergence implies that data measured by n_{TV} on communication channels between n_{TV} and other nodes is more complex than on other communication channels, and holds less distinctive features. Likely, this behaviour is caused by multipath signals due to the spacious area the node was located in. Thus, the location of a newly added client relative to other clients matters, which adds to the importance of distributed learning for HAR in the form of FL.

As mentioned, the removal of n_{AP} from the pre-training phase improved model performance, which suggests the presence of noisy data on this node. However, a higher convergence time than other nodes for the newly added n_{AP} as receiver is not observed. The opposite is true for n_{TV}. This suggests that multipath signals occurring on the receiving side of a communication link impact training performance less than when such signals occur close to the transmitter. This is intuitively explained by the fact that noisy signals close to a receiver are limited to being measured by that receiver only. On the other hand, noisy signals being caused close to the transmitter are measured by all other nodes in the system, contributing to considerably more noisy data.

The number of FL rounds executed during the pre-training phase affects the convergence rate of newly added receiving nodes. The negative learning rate for n_{TV} with 1000 pre-trained rounds is highly interesting and may be caused by the relatively high number pre-trained epochs and the noisy nature of the data from n_{TV}. It is plausible that the pre-trained model was overfitting to the non-noisy data, therefore requiring adjustments to the network weights to allow further learning on the new noisy data.

Lastly, the performance of various receiver nodes is indicated in Figure 7.19. If device resources allow, it is advised to perform a greater amount of pre-training to achieve better performance.

8.5.3 New client - transmitter

The observation that the addition n_{AP} results in longer convergence time further supports the claim that multipath signals are generated on the transmitter side of the communication link between n_{AP} and the other nodes. Furthermore, since the addition of n_{TV} as transmitter results in the highest performance for most training configurations, it is implied that signals originating from this node result in the most information-dense data. It can only be speculated why this is the case, and further research is needed to conclude on optimal device placement.

8.5.4 New client - complete network

In this last configuration, all the data that comes with the introduction of a fully operating device are considered. The relatively high performance of n_{TV} is in line with its relative performance when added as transmitter. The results mainly indicate that it is feasible to train existing models on unseen data and to adjust to new environments within less time than training a new model from scratch. This has already been proven by Hernandez et al. [5], and now this prove has been extended to activities in new locations and more dynamic transceiver pairs, allowing for flexibility in realistic environments.

8.5.5 Performance with limited data

As with the *location* set, availability of training data influences the performance and convergence time of the continuation of training based on pre-trained models. The trade-off between performance, convergence time, and data fraction is the same for the receiver, transmitter and complete setup, indicating the possibility of training with a little amount of data for each type of device (i.e. receiver or transmitter) as long as it is able to communicate in some manner.

8.6 Summary of Observations

A summarised overview of the most important observations is given in this section. The section aims to explicitly answer the research questions as stated in Chapter 1, thus the subjects considered are based on these questions: the benefits of FL, maximising the performance of FL, the impact of unseen data, and the proposed methods to deal with resource constraints. The summary for each subject begins by restating the relevant research question.

8.6.1 Benefits of federated learning

How does distributed computing using federated learning improve indoor human activity recognition on IoT devices compared to centralised machine learning approaches?

The performance comparison between classical ML and FL indicates that the classical approach achieves convergence in less time compared to FL. The classical approach consistently outperforms FL in terms of F1-score for different numbers of epochs and various amount of locally performed epochs in the FL approach. It is expected that with more heterogeneously distributed data, FL outperforms classical ML, especially with a fitting aggregation algorithm. With data that is IID among clients, FL does not offer significant improvement in convergence time, but it does enable clients to retain their data locally and share updates with the server. In contrast, the classical approach would require sharing all measured data, leading to increased communication overhead and resource consumption. Moreover, FL can be used in inter-household model sharing to enhance the ability of the classification to generalise, addressing privacy concerns associated with sharing CSI data among unfamiliar entities.

8.6.2 Maximising performance

Which federated learning settings can be used to maximise the performance of human activity recognition?

As discussed, it is generally true that a higher amount of local computation by clients results in a decrease in the convergence time. Model performance is increased as long as the amount of local computation is limited. In training continuation based on pre-trained models, the number of pre-trained epochs determines whether a higher amount of local computation is advantageous. Different aggregation algorithms affect the convergence time and model size, while the evaluation performance is similar for a relatively high amount of model training. Applying global momentum locally at clients (`MimeLite`) can be beneficial with an adaptive learning rate.

8.6.3 Unseen data

New location

How can a change in activity location be dealt with to create a location-independent federated learning system for human activity recognition in dynamic environments?

A comparison has been made between the training of existing models on data from various unseen locations. The performance of the models is found to be similar for each omitted location, but worse than the benchmark model due to reduced training data variety. When new locations are added, more local computation leads to faster convergence, but the wall time until convergence may not necessarily be reduced. Pre-training the models adequately improves the F1-score and reduces convergence time when new data are added. A change in activity location can thus be dealt with in a reliable manner by using existing models and allowing devices to use (part of) the new data to optimise for the new location, within the available time and resources. The specific requirements of different applications determine the extent of pre- and re-training possible and necessary.

New client

How can dynamic client participation be dealt with in federated learning for human activity recognition to ensure sufficient classification performance in dynamic environments?

Training is also performed on data from unseen clients. The findings reveal observations about different clients and their roles in the network. Models trained without client n_{AP} consistently outperform the benchmark score, indicating that the data measured by this client are noisy due to its placement at a corner with multipath signals. The newly introduced client n_{TV} as a receiver takes longer to converge during training, suggesting that the data it measures on communication channels with other nodes are more complex and less distinctive. The location of the new client relative to others plays a crucial role, highlighting the importance of distributed learning. When added as a transmitter, certain clients require longer convergence times, whereas others provide the most information-dense data and thus allow for a more rapid adaptation to the new data. Overall, the results demonstrate that training existing models on unseen data and adapting to new environments is feasible and efficient, allowing for dealing with dynamic client participation. The available time and resources determine performance and convergence time of the new network. Further research is needed to optimise device placement and draw more conclusive findings.

8.6.4 Resource constraints

What trade-offs exist between model complexity, classification performance, and resource constraints in human activity recognition for IoT devices and how can these trade-offs be optimised?

Memory limitations

In indoor HAR with the use of edge devices in an FL setting, the memory capacity of devices is limited. Therefore, this work has considered a small neural network with only three dense layers, which has proven to work in CSI-based HAR, as also indicated in the work of Hernandez et al. [5]. Additionally, both the batch size and number of locally performed epochs affect convergence time, while requiring an increase in available memory. Deciding on which settings to adopt depends on the resource limitations and the application requirements.

Convergence time

Convergence time becomes particularly crucial in network optimisation when resource constraints are present. Limited resources such as memory capacity and energy require optimising convergence time to ensure efficient utilisation of available resources. In such scenarios, the balance between achieving the desired performance and minimising the convergence time is key. It has been shown that convergence time can be limited through varying the batch size, the amount of local computation, the type averaging algorithm, the amount of data availability and the amount of pre-training in the case of unseen data.

Limited data

Limited data availability affects performance and convergence time, with the use of more data resulting in faster convergence, but equivalent performance is achieved with 50% of the available data. The trade-off between performance, convergence speed, and the amount of data considered during optimisation should be considered based on memory and computational limitations.

8.7 Limitations

The work in this research has been conducted to the best extent possible regarding resources and knowledge. However, shortcomings as a result of limitations of the work done during this thesis are present. This section highlights known limitations and briefly elaborates on their impact on the research.

8.7.1 Limited data gathering

The research was constrained in conducting more extensive data gathering due to the significant time required for each person to do so. In the ideal case, each subject would have conducted each activity in every location L_{TV}, L_{table} and L_{kitch}, with and without interference in the room for each transmitting device. This would have resulted in data for all possible variable combinations, allowing for narrowing down impacts on the data by individual variables in more detail. The number of measurements rapidly grows by doing so, which would have required a whole day of measurements for each individual.

In this study, the measurements were conducted with the aim of gathering a substantial amount of independent data. However, due to predetermined combinations of the main activity and randomly assigned interference activities, it was not feasible to determine the optimal location between the performed activity and the transmitter and receiver for achieving the highest classification accuracy, as was done in the pilot study.

8.7.2 TensorFlow implementation

To allow for convenient implementation and modification of an FL system, the TFF framework was used. This framework proved to be useful, but some limitations were encountered.

First, the implementation lacked an activity regulariser (L_1) in the FL framework. While it was shown that performance is acceptable without L_1 regularisation, it can be fruitful to include it in more complex data.

Second, the use of the framework solely allowed for FL simulation without considering communication overhead and device constraints. With a higher amount of communication, the cost that comes with it becomes increasingly important. The lack of it during this research might limit the real-world applicability and how generally applicable the findings are.

Third, the averaging algorithms that have been compared to each other in this work are only a subset of the algorithms that have been created in the current literature, as indicated in Chapter 3. The TFF framework only provides convenient implementation for the algorithms considered in this work. While it is possible to develop custom aggregation algorithms, resource limitations have prevented their inclusion in this study, leaving them as potential areas for future work. Next to that, decentralised approaches without a central server have been explored in Chapter 3, yet have not been implemented in this work due to TFF limitations.

Fourth, an implementation error led to the absence of deterministic random shuffling of the data. As a result, the analysis of different variables was conducted using randomly ordered data, possibly leading to inconsistent model performance of the models. To mitigate the negative impact, multiple runs of model optimisation were executed under different configurations.

Finally, the current TFF framework can only be used for research purposes and is not ready for implementation in the real world, as deployment to devices is nonexistent [88]. This limits the possibility of verifying the findings of this study.

8.7.3 Limited parameter exploration

The research focused on isolating specific variables to measure their individual influence, rather than analysing various combinations of parameters. Consequently, the exploration of parameter combinations was not included in the study, such as batch size with different averaging algorithms, or different algorithms with unseen data. This was done to limit the amount of time and resources necessary to come to the findings presented in this thesis.

8.7.4 Unseen data

The introduction of unseen data was simulated by removing part of the data from the training dataset after measurement and processing. This means that data was collected simultaneously with all other data, but intentionally withheld during the pre-training phase. Consequently, the supposedly "new" data are actually data from the past which the system was not allowed to access until training continuation on these data began. It is important to acknowledge that this approach may introduce biases, as it contains data from activities measured by other devices, as well.

8.7.5 Device presence

The research presented in this thesis assumes that indoor environments contain a sufficient number of IoT devices to establish an FL framework in which data are measured between them. However, IoT devices might not be readily available in large quantities, especially in elderly homes. In such cases, the advantage of not being required to install new hardware for CSI-based sensing disappears, thus smart homes are to be the norm in order for achieve the functionality of this system.

8.8 Ethical Consequences

With the realisation of CSI-based HAR, ethical concerns are introduced. These concerns and their consequences are highlighted in this section.

While data are stored locally without being shared, possibilities in exploiting model updates still remain. Private data could be reconstructed based on changes in the global model, introducing the need to randomise part of the global model [21].

The CSI data measured and gathered to perform HAR are on itself privacy sensitive, as it includes data on what a person is doing, as well as their heart rates and other vital signs [91]. When Wi-Fi signals from neighbouring environments are measured, privacy can be easily violated. Efforts should be made to ensure such data are not available to wrongful parties.

Lastly, as opposed to wearable devices or other observation methods, CSI data can be measured without the need to introduce new hardware to an environment. Therefore, a person's activity can be monitored without their knowledge and consent, which is concerning in most situations. Actions are to be taken to ensure a person being tracked is always aware of it.

Chapter 9

Conclusion and Future Work

First, the research question is restated again and the conclusion of this work is drawn. Then, recommendations for future work are stated.

The research question as stated in Chapter 1 is as follows:

> *How is CSI-based human activity recognition improved with the use of federated learning approaches, aimed towards practical use in changing environments with changing activity locations and client participation?*

FL improves indoor HAR compared to classical ML regarding privacy increase and communication reduction, but performance results indicate the preference of classical ML for indoor HAR. When FL is used, the classification performance is optimised through careful parameter tuning, deciding on batch size, the optimal amount of local computation, and finetuning the used aggregation algorithm. Training continuation has shown to provide proper classification models when personalising on unseen data, allowing for dynamic environments with changing activity location and device participation. Using edge or IoT devices introduces resource constraints, which necessitates the need for simple classification models. This work has shown that adequate performance can be achieved within reasonable time and limited data availability, resulting in communication-, energy- and memory-saving implementations. Specific settings affecting device resources are application-dependent and need to be adjusted according to each device's capabilities. Overall, the use of FL improves indoor HAR by allowing computational efforts from distributed devices and effectively adapting to new environments while respecting limitations resulting from individual device constraints.

9.1 Recommendations for Future Work

A physical implementation of FL for indoor HAR based on CSI data to monitor individuals inside their home seems feasible, but future work is necessary to achieve that. This section highlights possible interesting areas of focus.

First, applying the FL framework into a real-life environment with live classification using on-board computation is highly recommended. Doing so could highlight drawbacks and limitations of this research which have not occurred during simulations. In order for it to function properly, it is recommended to focus on communication protocols between clients and the server, since this and other works in FL for HAR have mostly simulated all forms of communication. Since different types of devices could contribute to the federated system, batch sizes might vary per device for optimal learning rate, as performed by Ma et al. [90].

A greater number of participating clients that measure CSI data in more than one indoor environment is recommended. This could result in more non-IID data, thus revealing how different aggregation algorithms compare to each other under such circumstances. By using a framework with more clients and environments, also more elaborate results on dynamic environments can be achieved, which could result in more explicit conclusions on the influence of activity location and device participation. Additionally, as done in the *WiFederated* framework, it might be useful to explore the effect of a varying number of participating clients for a dynamic environment in order to establish the required device presence.

It is advised to perform a detailed signal propagation analysis of indoor environments to conclude how multipath interference occurs due to device placement. In the results discussed in this work, only speculations could be made as of why certain activity or client locations resulted in better or worse performance results. By performing the signal analysis, more detailed conclusions can be drawn, resulting in more optimal device placement.

The implementation of more aggregation algorithms is encouraged, especially in implementations with more clients and potentially non-IID data among them. As mentioned, an FL implementation for HAR using `MimeLite` with momentum and adaptive learning rate could be investigated. Additionally, while decentralised FL algorithms exist, they have not been applied in this work or other HAR frameworks. It is advised to pursue a comparison between centralised and decentralised implementations, highlighting the differences in computational and communication costs. This is important in order to avoid the limitations caused by having a network with a single point of failure. Furthermore, a decentralised FL system would allow for a framework in which CSI data are transmitted interchangeably by different clients for a short period of time. By doing so, the influence of the activity location relative to the transmitter is mitigated, as observations can be made from different angles.

Besides changing the network topology, it is recommended to analyse the effect of quantisation methods on model size and performance. For this work, no such methods were implemented, but it has been shown that quantisation can a provide model reduction in FL while compromising only by a small amount on the classification performance [92].

Lastly, it is recommended to perform a study regarding the ethical consequences as mentioned in Section 8.8. It should be analysed how the CSI data or model updates communicated by a client can be exploited. Privacy concerns must be examined and addressed before HAR using FL can be employed in real-world settings.

Bibliography

[1] P. V. Zahorodko, S. O. Semerikov, V. N. Soloviev, A. M. Striuk, M. I. Striuk, and H. M. Shalatska, “Comparisons of performance between quantum-enhanced and classical machine learning algorithms on the ibm quantum experience,” *Journal of Physics: Conference Series*, vol. 1840, no. 1, p. 012021, mar 2021. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1840/1/012021

[2] T. Q. Dinh, D. N. Nguyen, D. T. Hoang, T. V. Pham, and E. Dutkiewicz, “In-network Computation for Large-scale Federated Learning over Wireless Edge Networks,” *IEEE Transactions on Mobile Computing*, pp. 1–15, 2022.

[3] J. Liu, H. Liu, Y. Chen, Y. Wang, and C. Wang, “Wireless Sensing for Human Activity: A Survey,” *IEEE Communications Surveys and Tutorials*, vol. 22, no. 3, pp. 1629–1645, 2020.

[4] S. K. Yadav, S. Sai, A. Gundewar, H. Rathore, K. Tiwari, H. M. Pandey, and M. Mathur, “CSITime: Privacy-preserving human activity recognition using WiFi channel state information,” *Neural Networks*, vol. 146, pp. 11–21, 2022.

[5] S. M. Hernandez and E. Bulut, “WiFederated: Scalable WiFi Sensing using Edge Based Federated Learning,” *IEEE Internet of Things Journal*, vol. 9, no. 14, pp. 12 628–12 640, 2021.

[6] K. Sozinov, V. Vlassov, and S. Girdzijauskas, “Human activity recognition using federated learning,” *Proceedings - 16th IEEE International Symposium on Parallel and Distributed Processing with Applications, 17th IEEE International Conference on Ubiquitous Computing and Communications, 8th IEEE International Conference on Big Data and Cloud Computing, 11t*, no. 2, pp. 1103–1111, 2019.

[7] L. Tu, X. Ouyang, J. Zhou, Y. He, and G. Xing, “FedDL: Federated Learning via Dynamic Layer Sharing for Human Activity Recognition,” *SenSys 2021 - Proceedings of the 2021 19th ACM Conference on Embedded Networked Sensor Systems*, pp. 15–28, 2021.

[8] A. Fallah, A. Mokhtari, and A. E. Ozdaglar, “Personalized federated learning: A meta-learning approach,” *CoRR*, vol. abs/2002.07948, 2020. [Online]. Available: https://arxiv.org/abs/2002.07948

[9] A. Bellet, A. Kermarrec, and E. Lavoie, “D-cliques: Compensating noniidness in decentralized federated learning with topology,” *CoRR*, vol. abs/2104.07365, 2021. [Online]. Available: https://arxiv.org/abs/2104.07365

[10] “Federated learning: Collaborative machine learning without centralized training data,” https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, Apr 2017, accessed on 30-09-2022.

[11] C. Aggarwal, *Neural Networks and Deep Learning: A Textbook.* Springer International Publishing, 2018.

[12] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, 2006.

[13] J. Brownlee, “A gentle introduction to the rectified linear unit (relu),” https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/, accessed on 15-05-2023.

[14] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On large-batch training for deep learning: Generalization gap and sharp minima,” 2017.

[15] A. Oppermann, “Regularization in deep learning—l1, l2, and dropout | towards data science,” https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036, 2 2020.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.

[17] Keras, “Layer weight regularizers,” https://keras.io/api/layers/regularizers/, accessed on 19-12-2022.

[18] Z. Liu, Z. Xu, J. Jin, Z. Shen, and T. Darrell, “Dropout reduces underfitting,” *arXiv preprint arXiv:2303.01500*, 2023.

[19] K. P. Shung, “Accuracy, precision, recall or f1?” https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9, accessed on 28-02-2023.

[20] Google Developers, “Classification: Precision and recall, machine learning crash course,” https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall, accessed on 28-02-2023.

[21] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. D’Oliveira, H. Eichner, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecní, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and

S. Zhao, “Advances and open problems in federated learning,” *Foundations and Trends in Machine Learning*, vol. 14, no. 1-2, pp. 1–210, 2021.

[22] S. W. Remedios, J. A. Butman, B. A. Landman, and D. L. Pham, “Federated Gradient Averaging for Multi-Site Training with Momentum-Based Optimizers,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12444 LNCS, pp. 170–180, 2020.

[23] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, “Communication-efficient learning of deep networks from decentralized data,” *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, vol. 54, 2017.

[24] W. Liu, L. Chen, and W. Zhang, “Decentralized Federated Learning: Balancing Communication and Computing Costs,” *IEEE Transactions on Signal and Information Processing over Networks*, vol. 8, pp. 131–143, 2022.

[25] C. Hu, J. Jiang, and Z. Wang, “Decentralized federated learning: A segmented gossip approach,” *CoRR*, vol. abs/1908.07782, 2019. [Online]. Available: http://arxiv.org/abs/1908.07782

[26] X. Lian, C. Zhang, H. Zhang, C. J. Hsieh, W. Zhang, and J. Liu, “Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent,” in *Advances in Neural Information Processing Systems*, vol. 2017-Decem, 2017, pp. 5331–5341.

[27] Z. Jiang, A. Balu, C. Hegde, and S. Sarkar, “Collaborative deep learning in fixed topology networks,” in *Advances in Neural Information Processing Systems*, vol. 2017-Decem, 2017, pp. 5905–5915.

[28] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, “On the Convergence of FedAvg on Non-IID Data,” *arXiv preprint arXiv:1907.02189*, no. 2019, pp. 1–26, 2019. [Online]. Available: http://arxiv.org/abs/1907.02189

[29] Z. Chen, L. Zhang, C. Jiang, Z. Cao, and W. Cui, “Wifi csi based passive human activity recognition using attention based blstm,” *IEEE Transactions on Mobile Computing*, vol. 18, no. 11, pp. 2714–2724, 2019.

[30] M. Kotaru, K. Joshi, D. Bharadia, and S. Katti, “SpotFi: Decimeter Level Localization Using WiFi,” *Computer Communication Review*, vol. 45, no. 4, pp. 269–282, 2015.

[31] P. F. Moshiri, H. Navidan, R. Shahbazian, S. A. Ghorashi, and D. Windridge, “Using gan to enhance the accuracy of indoor human activity recognition,” 2020. [Online]. Available: https://arxiv.org/abs/2004.11228

[32] A. Zhuravchak, O. Kapshii, and E. Pournaras, “Human Activity Recognition based on Wi-Fi CSI Data -A Deep Neural Network Approach,” *Procedia*

Computer Science, vol. 198, no. 2021, pp. 59–66, 2021. [Online]. Available: https://doi.org/10.1016/j.procs.2021.12.211

[33] R. Khusainov, D. Azzi, I. E. Achumba, and S. D. Bersch, “Real-time human ambulation, activity, and physiological monitoring: Taxonomy of issues, techniques, applications, challenges and limitations,” *Sensors*, vol. 13, no. 10, pp. 12 852–12 902, 2013. [Online]. Available: https://www.mdpi.com/1424-8220/13/10/12852

[34] A. Khan, N. Hammerla, S. Mellor, and T. Plötz, “Optimising sampling rates for accelerometer-based human activity recognition,” *Pattern Recognition Letters*, vol. 73, pp. 33–40, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2016.01.001

[35] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, “Adaptive Federated Learning in Resource Constrained Edge Computing Systems,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[36] J. Mao, H. Yang, P. Qiu, J. Liu, and A. Yener, “Charles: Channel-quality-adaptive over-the-air federated learning over wireless networks,” 2022. [Online]. Available: https://arxiv.org/abs/2205.09330

[37] T. Nishio and R. Yonetani, “Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge,” *IEEE International Conference on Communications*, vol. 2019-May, 2019.

[38] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous metworks,” *CoRR*, vol. abs/1812.06127, 2018. [Online]. Available: http://arxiv.org/abs/1812.06127

[39] F. Sattler, S. Wiedemann, K. R. Muller, and W. Samek, “Robust and Communication-Efficient Federated Learning from Non-i.i.d. Data,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, 2020.

[40] T. Li, M. Sanjabi, and V. Smith, “Fair resource allocation in federated learning,” *CoRR*, vol. abs/1905.10497, 2019. [Online]. Available: http://arxiv.org/abs/1905.10497

[41] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, “SCAFFOLD: Stochastic Controlled Averaging for Federated Learning,” *37th International Conference on Machine Learning, ICML 2020*, vol. PartF16814, pp. 5088–5099, 2020.

[42] S. P. Karimireddy, M. Jaggi, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, “Mime: Mimicking centralized stochastic algorithms in federated learning,” *arXiv: Learning*, 2021.

[43] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *CoRR*, vol. abs/1610.05492, 2016. [Online]. Available: http://arxiv.org/abs/1610.05492

[44] J. Li, Y. Shao, K. Wei, M. Ding, C. Ma, L. Shi, Z. Han, and H. V. Poor, "Blockchain Assisted Decentralized Federated Learning (BLADE-FL): Performance Analysis and Resource Allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2401–2415, 2022.

[45] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Communications Letters*, vol. 24, no. 6, pp. 1279–1283, 2020.

[46] A. Hashemi, A. Acharya, R. Das, H. Vikalo, S. Sanghavi, and I. Dhillon, "On the Benefits of Multiple Gossip Steps in Communication-Constrained Decentralized Federated Learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2727–2739, 2022.

[47] X. Li, W. Yang, S. Wang, and Z. Zhang, "Communication-Efficient Local Decentralized SGD Methods," *arXiv preprint arXiv:1910.09126*, vol. 14, no. 8, pp. 1–20, 2019. [Online]. Available: http://arxiv.org/abs/1910.09126

[48] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar, "MATCHA: Speeding Up Decentralized SGD via Matching Decomposition Sampling," in *2019 6th Indian Control Conference, ICC 2019 - Proceedings*, 2019, pp. 299–300.

[49] J. Wang and G. Joshi, "Cooperative SGD: A unified Framework for the Design and Analysis of Communication-Efficient SGD Algorithms," *CoRR*, vol. abs/1808.0, 2018. [Online]. Available: http://arxiv.org/abs/1808.07576

[50] F. P. C. Lin, S. Hosseinalipour, S. S. Azam, C. G. Brinton, and N. Michelusi, "Semi-Decentralized Federated Learning with Cooperative D2D Local Model Aggregations," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3851–3869, 2021.

[51] "The cifar-10 dataset," https://www.cs.toronto.edu/~kriz/cifar.html.

[52] "The mnist database," http://yann.lecun.com/exdb/mnist/.

[53] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: an extension of mnist to handwritten letters," 2017.

[54] Z. Research, "Fashion mnist," https://www.kaggle.com/datasets/zalando-research/fashionmnist, Dec 2017.

[55] S. Caldas, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018. [Online]. Available: https://arxiv.org/abs/1812.01097

[56] Y. Wang, X. Jiang, R. Cao, and X. Wang, "Robust indoor human activity recognition using wireless signals," *Sensors (Switzerland)*, vol. 15, no. 7, pp. 17 195–17 208, 2015.

[57] W. Wang, A. X. Liu, M. Shahzad, K. Ling, and S. Lu, "Device-Free Human Activity Recognition Using Commercial WiFi Devices," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1118–1131, 2017.

[58] V. Smith, C. K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 4425–4435, 2017.

[59] W. Zhang, Z. Wang, and X. Wu, "WiFi Signal-Based Gesture Recognition Using Federated Parameter-Matched Aggregation," *Sensors*, vol. 22, no. 6, pp. 1–14, 2022.

[60] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A Survey on Federated Learning for Resource-Constrained IoT Devices," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1–24, 2022.

[61] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, and S. Avestimehr, "Federated learning for internet of things: Applications, challenges, and opportunities," 2022.

[62] O. A. Wahab, A. Mourad, H. Otrok, and T. Taleb, "Federated Machine Learning: Survey, Multi-Level Classification, Desirable Criteria and Future Directions in Communication and Networking Systems," *IEEE Communications Surveys and Tutorials*, vol. 23, no. 2, pp. 1342–1397, 2021.

[63] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 8114–8124, 2019.

[64] J. Brownlee, "Gradient descent with momentum from scratch," https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch/, accessed on 19-05-2023.

[65] N. D. Fabbro, S. Dey, M. Rossi, and L. Schenato, "A Newton-type algorithm for federated learning based on incremental Hessian eigenvector sharing," *arXiv preprint arXiv:2202.05800*, pp. 1–46, 2022. [Online]. Available: http://arxiv.org/abs/2202.05800

[66] H. Xing, O. Simeone, and S. Bi, "Decentralized Federated Learning via SGD over Wireless D2D Networks," *IEEE Workshop on Signal Processing Advances in Wireless Communications, SPAWC*, vol. 2020-May, 2020.

[67] Y. Liu, S. Garg, J. Nie, Y. Zhang, Z. Xiong, J. Kang, and M. S. Hossain, "Deep Anomaly Detection for Time-Series Data in Industrial IoT: A Communication-Efficient On-Device Federated Learning Approach," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6348–6358, 2021.

[68] J. Xu, W. Du, Y. Jin, W. He, and R. Cheng, "Ternary Compression for Communication-Efficient Federated Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 3, pp. 1162–1176, 2022.

[69] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. U. Stich, “A Unified Theory of Decentralized SGD with Changing Topology and Local Updates,” in *37th International Conference on Machine Learning, ICML 2020*, vol. PartF16814, 2020, pp. 5337–5349.

[70] K. I. Tsianos, S. F. Lawlor, and M. G. Rabbat, “Communication/computation tradeoffs in consensus-based distributed optimization,” *CoRR*, vol. abs/1209.1076, 2012. [Online]. Available: http://arxiv.org/abs/1209.1076

[71] Z. Zhao, J. Xia, L. Fan, X. Lei, G. K. Karagiannidis, and A. Nallanathan, “System Optimization of Federated Learning Networks With a Constrained Latency,” *IEEE Transactions on Vehicular Technology*, vol. 71, no. 1, pp. 1095–1100, 2022.

[72] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, “Inverting gradients - How easy is it to break privacy in federated learning?” in *Advances in Neural Information Processing Systems*, vol. 2020-Decem, no. 1, 2020, pp. 1–23.

[73] “Intel® Wi-Fi 6E AX211 (Gig+) Module,” https://www.intel.com/content/www/us/en/products/docs/wireless/wi-fi-6e-ax211-module-brief.html.

[74] “PicoScenes: Supercharging Your Next Wi-Fi Sensing Research!” https://ps.zpj.io/{#}, accessed on 17-11-2022.

[75] “Propagation revisited: Wireless multipath,” https://www.controleng.com/articles/propagation-revisited-wireless-multipath/, accessed on 13-04-2023.

[76] University of Twente, Techmed Centre, “Living lab ehealth house,” https://www.utwente.nl/en/techmed/facilities/htwb-labs/ehealth-house/, accessed on 6-12-2022.

[77] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, “Tool release: Gathering 802.11n traces with channel state information,” *ACM SIGCOMM CCR*, vol. 41, no. 1, p. 53, Jan. 2011.

[78] “Intel Ultimate N WiFi Link 5300,” https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ultimate-n-wifi-link-5300-brief.pdf, accessed on 30-03-2023.

[79] S. Raubitzek and T. Neubauer, “A fractal interpolation approach to improve neural network predictions for difficult time series data,” *Expert Systems with Applications*, vol. 169, 5 2021.

[80] “Linear Interpolation Formula,” https://www.cuemath.com/linear-interpolation-formula/, accessed on 06-04-2023.

[81] J. Klein Brinke, A. Chiumento, and P. Havinga, “Channel state information for human activity recognition with low sampling rates,” *UMUM 2023: Second Workshop on Ubiquitous and Multi-domain User Modeling*, pp. 614–620, 2023.

[82] "Everything you need to know about Min-Max normalization: A Python tutorial," https://towardsdatascience.com/everything-you-need-to-know-about-min-max-normalization-in-python-b79592732b79, accessed on 03-04-2023.

[83] "Standardization VS Normalization," https://dataakkadian.medium.com/standardization-vs-normalization-da7a3a308c64, accessed on 03-04-2023.

[84] "Rolling Averages: What They Are and How To Calculate Them," https://www.indeed.com/career-advice/career-development/what-is-rolling-average#:~:text=Rolling%20averages%20are%20useful%20for,might%20become%20difficult%20to%20track., accessed on 03-04-2023.

[85] "TFF for Federated Learning Research: Model and Update Compression," https://www.tensorflow.org/federated/tutorials/tff_for_federated_learning_research_compression, accessed on 31-01-2023.

[86] "Keras API reference, Layers API, Regularization layers, Dropout layer," https://keras.io/api/layers/regularization_layers/dropout/, accessed on 05-12-2022.

[87] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[88] "TensorFlow Federated: Machine Learning on Decentralized Data," https://www.tensorflow.org/federated, accessed on 31-01-2023.

[89] "tf.keras.metrics.CategoricalAccuracy," https://www.tensorflow.org/api_docs/python/tf/keras/metrics/CategoricalAccuracy, accessed on 06-02-2023.

[90] Z. Ma, Y. Xu, H. Xu, Z. Meng, L. Huang, and Y. Xue, "Adaptive batch size for federated learning in resource-constrained edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, pp. 37–53, 1 2023.

[91] X. Wang, C. Yang, and S. Mao, "On csi-based vital sign monitoring using commodity wifi," *ACM Trans. Comput. Healthcare*, vol. 1, no. 3, may 2020. [Online]. Available: https://doi.org/10.1145/3377165

[92] K. Gupta, M. Fournarakis, M. Reisser, C. Louizos, and M. Nagel, "Quantization robust federated learning for efficient inference on heterogeneous devices," 2022.

Appendices

A Overview of FL methods

Table A.1: Overview of summarised FL algorithms.

Study	Year	# citations	Summary
[35]	2019	995	Determine frequency of global aggregation to maximise efficiency of available resources.
[36]	2022	1	Adjust the power level and number of updates for clients based on the estimated CSI.
[23]	2016	6293	Basis of FL, taking the average of client updates for aggregation.
[37]	2019	705	Client selection to maximise aggregation, based on resource information like wireless channel states, computational capacities, and size of data resources.
[7]	2021	9	Group users with similar data distributions to share network layers.
[38]	2018	1278	Adds term to `FedAvg` penalising models that deviate from global model.
[2]	2022	2	Edge nodes between users and server aggregating model updates from users before sending update to central server.
[39]	2020	672	Framework implementing compression for communication from server to clients. Efficient with small number of clients.
[8]	2020	239	Train model such that users can personalise it with few updates.
[40]	2019	382	Penalising worse performing clients to bias towards clients with big local losses.
[41]	2020	578	Correct local update based on the difference between the global and local model.
[43]	2016	2773	Reduction of communication cost to server by learning an update using less variables & compression of model update.
[44]	2022	23	Method for decentralised FL using blockchain. Clients compete to determine who adds to the blockchain, avoiding separate miners and instead letting clients perform this task.
[45]	2020	346	Implements a distributed ledger in which local model updates are exchanged and verified, biased towards clients with more data available. Mining is done by dedicated miners.
[24]	2022	5	Multiple local update steps and multiple communication steps, with data compression.
[25]	2019	84	Segmented gossip approach where the model is communicated in segments. Clients receive segmented models which subsequently form an aggregated model.
[9]	2021	5	Cliques are formed in which the data distribution resembles the global data distribution.
[46]	2022	2	Implements compressed communication through gossiping.
[27]	2017	136	Decentralised SGD, with a single update and single communication step for each client.
[47]	2019	18	Allows varying amount of local update and SGD steps.
[48]	2019	96	Randomly selected clients perform computation and communication based on sparse subgraphs.
[49]	2018	276	Allows multiple local update steps.
[50]	2021	16	Combination of communication between clients and server, and between individual clients. A gradient descent step is performed at each client, communication then occurs within clusters.

[a] Accuracy tested with various open-access datasets: CIFAR-10 [51], MNIST [52], Fashion-MNIST [54], Synthetic, Vehicle, Sent140 & Shakespeare [55].
[b] Not mentioned in study.
[c] Depending on the type of data.
[d] Depending on topology, momentum & averaging methods.

B Expanded HAR literature overview

Table B.1: Expanded overview of related literature on human activity recognition.

Study	**Year**	**# citations**	**Summary**	**Recognition type**	**Classifier**	**Data used**
Robust Indoor Human Activity Recognition Using Wireless Signals [56]	2015	81	Performing HAR based in a non-FL setting.	Activity	SVM	Wi-Fi CSI
Device-Free Human Activity Recognition Using Commercial Wi-Fi Devices [57]	2017	317	HAR based in a non-FL setting, using commercially available devices.	Activity	HMM	Wi-Fi CSI
Federated Multi-Task Learning [58]	2018	1088	Multi-task FL to train personalised models. It allows clients to approximate the model update to avoid stragglers.	Activity	SVM	Sensor data from Google Glass, smartphones and vehicles
Human Activity Recognition Using Federated Learning [6]	2019	79	FL setting for HAR using sensors data.	Activity	DNN & softmax regression	Sensor data from smartphones and smartwatches
WiFederated: Scalable WiFi Sensing Using Edge-Based Federated Learning [5]	2022	6	FL setting for HAR. Aggregation is done with `FedAvg`.	Activity	DNN	Wi-Fi CSI
WiFi Signal-Based Gesture Recognition Using Federated Parameter-Matched Aggregation [59]	2022	0	FL setting for gesture recognition based on the body-coordinate velocity profile from CSI.	Gesture	CNN & RNN	Wi-Fi CSI

C Algorithm overview of centralised federated learning

Typical centralised FL frameworks operate by having clients send model updates to a central server. The server subsequently aggregates all the model updates via one of many different averaging methods before sending the newly updated model out to the clients. The manner of processing the individual model updates influences the accuracy and convergence performance of the overall model. Therefore, much research considering the improvement of FL has covered the influence of the averaging algorithm and how to improve it.

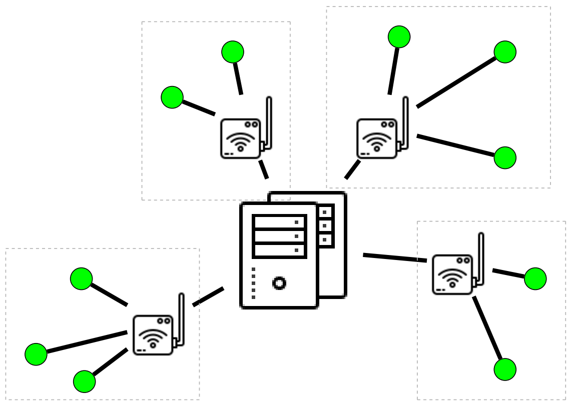


Figure C.1: A centralised topology with in-network computation [2], where nodes (green) from different rooms (grey borders) send model updates to a local access point before being sent to the central server.

- **FedProx**
  `FedProx` is based on `FedAvg` and introduces a proximal term to better deal with heterogeneous data. The method allows for a varying amount of epochs for each node before sending updates to the server. An update is influenced by the proximal term $\frac{\mu}{2}||w - w^t||^2$, with w the updated weights and w^t the weights received from the server. With this term, nodes are penalised when their weights deviate from the global model weights, with the hyperparameter μ to tune the influence of the penalty. This method enables different nodes to do different amounts of work, which could be useful when the amount of available data are different for each node [38, 60].

- **q-FedAvg**
  In `q-FedAvg`, the following loss-function $f_q(w)$ is being minimised:

$$f_q(w) = \sum_{i=1}^{m} \frac{p_i}{q+1} f_i^{q+1}(w), \tag{9.1}$$

  with m the total number of devices, p_i the probability of choosing device i for participating in the current round, and f_i the local cost function. Worse performing clients are penalised by a power of $q+1$, q being a tunable hyperparameter. This way, instead of biasing towards clients with more data points as in `FedAvg`, the model biases towards clients with bigger local losses. The end result is a model that performs more uniformly across clients and should thus be more fair [40].

- **Per-FedAvg**
  In `Per-FedAvg`, the loss function being minimised is based on the weights after a gradient descent step of the local function:

$$f(w) = \sum_{i=1}^{m} p_i f_i(w - \alpha \nabla f_i(w)), \tag{9.2}$$

  with $\alpha \geq 0$ the stepsize. The idea is to slightly update an initialised model based on a client's own data, which should result in a model specifically designed for each client [8].

- **SCAFFOLD**
  `FedAvg` used in non-IID data conditions results in client drift. As an alternative, `SCAFFOLD` (Stochastic Controlled Averaging algorithm) is introduced. This method works well when non-IID data are involved, since the difference in the update direction of the server and that of a client (so-called *control variates*) are used to correct the local update. Furthermore, `SCAFFOLD` takes similarity of data between clients into account, thus requiring less communication rounds when data across clients is similar. The algorithm does rely on clients being *stateful*, meaning that a client's model state is preserved across training rounds and thus the same clients are present in different rounds [41].

- **MimeLite**
  `MimeLite` also deals with client drift by applying global momentum locally at clients, reducing convergence time. As opposed to `SCAFFOLD`, the algorithm does not rely on stateful clients, since the momentum used in `MimeLite` depends on the aggregated client updates. Therefore, the algorithm allows for new clients every training round. The momentum parameter β influences the degree to which the previous update influences the current update [42].

- **In-network Computation**
  This method aims to reduce the amount of communication towards and computation at the central server, by having edge nodes collect model updates from users and aggregating these to form an aggregated model for the user partition under that edge node. This method can also be applied to form groups of nodes that operate in the same room as illustrated in Fig. C.1. Subsequently, the edge node sends the aggregated model to the central server where all models are aggregated to form the global model. The method allows for proper convergence while outperforming other methods. Moreover, the method mitigates straggler issues by letting nodes send their results whenever they are available [2]. Stragglers are nodes that take much longer to report an output (i.e. model update) than other nodes. As a consequence, the central server has to wait longer before performing the aggregation step.

- **FedDL**
  `FedDL` makes use of similarity of data between nodes to perform HAR. A deep learning network consisting of multiple layers is trained, where the lower layer of the network is shared by most users and captures general features. The top layers of the network represent features more specific to users. The model is thus created iteratively in a layer-wise manner. The end result is a personalised model for each client, clients that have access to data from similar data distributions share more network layers. Similarity between data distributions is learned by testing the local models against a reference distribution, which might cause insight in user activities for the central server. On the other hand, this method does reduce communication in the network [7].

D Algorithm overview of decentralised federated learning

Presented below is a selection of methods used in a decentralised FL setup. These and other methods focusing on decentralised FL are noted in Table 3.1.

- **PD-SGD**
  Periodic Decentralised SGD (`PD-SGD`) improves on `D-SGD` by allowing for multiple local update steps. Through such periodic averaging and only communicating to direct neighbours in the topology, communication is reduced [49]. However, the method is lacking a proper trade-off between communication and computation [47]. Moreover, the method is not adapted to non-IID data cases [21].

- **LD-SGD**
  Improves in `PD-SGD` by expanding the method to be suitable for non-IID data cases by allowing for a varying amount of local update steps and `D-SGD`'s (with an update step, communication step and aggregation step) [47].

- **MATCHA**
  As improvement on `PD-SGD`, `MATCHA` introduces an additional step where clients are randomly selected to perform computation and communication steps. Disjoint matching is executed on the starting topology, after which sparse subgraphs are created by activating part of the matchings. Only this subtopology is used during update steps, thus reducing communication. A subgraph that is more closely connected is favoured, thus being activated more often. The combination of these methods result in a faster convergence than `PD-SGD` [48]. Comparing `MATCHA` to `LD-SGD` gives rise to an important observation: `LD-SGD` generally has a higher training loss than `MATCHA`, yet `LD-SGD` reaches a test accuracy similar to `MATCHA` in less wall-clock time. It could be concluded from this that in the decentralised FL setting, increasing local computation is favoured over decreasing communication between clients. Again, a trade-off between communication and computation can be observed.

- **C-DFL**
  According to [24], improving communication efficiency in a decentralised FL setup while also considering a reasonably convergence time of a model consensus is not widely addressed. This work proposes a decentralised FL framework in which nodes alternately perform multiple local model updates, after which multiple communications between nodes occur. An additional method was also introduced which improves communication efficiency by means of data compression: `C-DFL`. Sparsification is used to decrease the amount of data to be sent to other nodes. This results in a convergence increase of up to 74.6%.

- **Combo**
  The `Combo` method is based on a segmented gossip approach. Clients first perform one or multiple local update steps and split the model into multiple segments. Segments are sent to randomly selected clients who aggregate the model based on all received

segments. This method make efficient use of the network in a decentralised setting. By sending out only segments of the model the entire bandwidth of communication channels between clients is used. This is particularly useful when clients are located far from each other, and thus when there exist links that do not allow clients to use the full bandwidth. The model converges in less time than the standard gossiping approach and reaches similar accuracy levels.
An additional benefit of `Combo` is that pulling requests for new model segments can be cancelled when the target client can not be reached. This allows for dynamic addition and removal of clients [25].

- **D-cliques**
  In `D-cliques`, groups of nodes are made to ensure every node-group (clique) contains data with a similar label distribution. Because of this, a sparse inter-clique topology can be used and nodes can keep communication to within a clique, thus reducing communication within the whole network. The method maintains similar accuracy under non-IID data settings to topologies in which nodes are all connected to each other, without slowing down the convergence. `D-cliques` is relatively scalable as it reaches acceptable accuracy with up to 1000 nodes under certain network topologies. A drawback of `D-cliques` is nodes being required to share the label distribution of their data with other nodes. This is in contrast to a typical FL system where information on the data of a node is held private [9].

- **DeLi-CoCo**
  Communication compression and gossip steps. After a local update step through gradient descent, 1 or more compressed communication steps via gossiping are performed. The work shows that more communication steps allow for faster convergence with certain compression rates. In a non-IID data setting, client drift occurs and the model does not reach a desirable test accuracy for complex datasets such as CIFAR-10 [46].

E Experiment schedule pilot study

Table E.1: Experiment schedule pilot study.

Time (min.)	#	Activity	Who?	Node configuration	Interference
15:00	1	Stand up & sit down	1	TV node transmitter	No
17:30	2	Stand up & sit down	0	TV node transmitter	No
20:00	3	Working	1	TV node transmitter	No
22:30	4	Working	0	TV node transmitter	No
25:00:00	5	Eat & drink	1	TV node transmitter	No
27:30:00	6	Eat & drink	0	TV node transmitter	No
30:00:00	7	Walk around	1	TV node transmitter	No
32:30:00	8	Walk around	0	TV node transmitter	No
35:00:00	9	Stand up & sit down	1	TV node transmitter	Yes
37:30:00	10	Stand up & sit down	0	TV node transmitter	Yes
40:00:00	11	Working	1	TV node transmitter	Yes
42:30:00	12	Working	0	TV node transmitter	Yes
45:00:00	13	Eat & drink	1	TV node transmitter	Yes
47:30:00	14	Eat & drink	0	TV node transmitter	Yes
50:00:00	15	Walk around	1	TV node transmitter	Yes
52:30:00	16	Walk around	0	TV node transmitter	Yes
55:00:00	17	Stand up & sit down	1	TV node receiver	No
57:30:00	18	Stand up & sit down	0	TV node receiver	No
60:00:00	19	Working	1	TV node receiver	No
62:30:00	20	Working	0	TV node receiver	No
65:00:00	21	Eat & drink	1	TV node receiver	No
67:30:00	22	Eat & drink	0	TV node receiver	No
70:00:00	23	Walk around	1	TV node receiver	No
72:30:00	24	Walk around	0	TV node receiver	No
75:00:00	25	Stand up & sit down	1	TV node receiver	Yes
77:30:00	26	Stand up & sit down	0	TV node receiver	Yes
80:00:00	27	Working	1	TV node receiver	Yes
82:30:00	28	Working	0	TV node receiver	Yes
85:00:00	29	Eat & drink	1	TV node receiver	Yes
87:30:00	30	Eat & drink	0	TV node receiver	Yes
90:00:00	31	Walk around	1	TV node receiver	Yes
92:30:00	32	Walk around	0	TV node receiver	Yes

F Experiment schedule eHealth House

The following tables present the experiment schedule for the eHealth House experiments for three different sets, in which the main activity (i.e. activity to be classified) is different for each set (P_0 for set 0 located at L_{tv}, P_1 for set 1 located at L_{table}, and P_2 for set 2 located at L_{kitch}).

In the tables, the following node numbering is used:

- Node 0: n_{kitch}
- Node 2: n_{table}
- Node 3: n_{ap}
- Node 4: n_{tv}

The activities are labeled as follows:

- A_0: Sit/stand
- A_1: Eat/drink
- A_2: Work
- A_3: Rest

Table F.1: Schedule eHealth House experiments, set 0.

Time	**Transmitter node**	**Activity P_0**	**Activity P_1**	**Activity P_2**	**File name**
00:10:00	0	A_0	A_3	A_1	tv_A0_T0
00:13:20	2	A_0	A_3	A_3	tv_A0_T2
00:16:40	3	A_0	A_3	A_0	tv_A0_T3
00:20:00	4	A_0	A_1	A_0	tv_A0_T4
00:23:20	0	A_1	A_1	A_3	tv_A1_T0
00:26:40	2	A_1	A_0	A_2	tv_A1_T2
00:30:00	3	A_1	A_0	A_1	tv_A1_T3
00:33:20	4	A_1	A_1	A_1	tv_A1_T4
00:36:40	0	A_2	A_2	A_1	tv_A2_T0
00:40:00	2	A_2	A_1	A_3	tv_A2_T2
00:43:20	3	A_2	A_2	A_2	tv_A2_T3
00:46:40	4	A_2	A_3	A_3	tv_A2_T4
00:50:00	0	A_3	A_2	A_1	tv_A3_T0
00:53:20	2	A_3	A_2	A_2	tv_A3_T2
00:56:40	3	A_3	A_1	A_2	tv_A3_T3
01:00:00	4	A_3	A_0	A_2	tv_A3_T4

Table F.2: Schedule eHealth House experiments, set 1.

Time	**Transmitter node**	**Activity P_0**	**Activity P_1**	**Activity P_2**	**File name**
00:10:00	0	A_2	A_0	A_0	table_A0_T0
00:13:20	2	A_3	A_0	A_2	table_A0_T2
00:16:40	3	A_0	A_0	A_0	table_A0_T3
00:20:00	4	A_1	A_0	A_3	table_A0_T4
00:23:20	0	A_2	A_1	A_2	table_A1_T0
00:26:40	2	A_2	A_1	A_0	table_A1_T2
00:30:00	3	A_1	A_1	A_3	table_A1_T3
00:33:20	4	A_3	A_1	A_3	table_A1_T4
00:36:40	0	A_1	A_2	A_3	table_A2_T0
00:40:00	2	A_3	A_2	A_0	table_A2_T2
00:43:20	3	A_0	A_2	A_2	table_A2_T3
00:46:40	4	A_0	A_2	A_3	table_A2_T4
00:50:00	0	A_0	A_3	A_0	table_A3_T0
00:53:20	2	A_2	A_3	A_2	table_A3_T2
00:56:40	3	A_2	A_3	A_1	table_A3_T3
01:00:00	4	A_2	A_3	A_3	table_A3_T4

Table F.3: Schedule eHealth House experiments, set 2.

Time	**Transmitter node**	**Activity P_0**	**Activity P_1**	**Activity P_2**	**File name**
00:10:00	0	A_1	A_1	A_0	kitch_A0_T0
00:13:20	2	A_3	A_0	A_0	kitch_A0_T2
00:16:40	3	A_1	A_0	A_0	kitch_A0_T3
00:20:00	4	A_0	A_3	A_0	kitch_A0_T4
00:23:20	0	A_0	A_1	A_1	kitch_A1_T0
00:26:40	2	A_2	A_3	A_1	kitch_A1_T2
00:30:00	3	A_1	A_1	A_1	kitch_A1_T3
00:33:20	4	A_1	A_3	A_1	kitch_A1_T4
00:36:40	0	A_3	A_3	A_2	kitch_A2_T0
00:40:00	2	A_3	A_1	A_2	kitch_A2_T2
00:43:20	3	A_1	A_3	A_2	kitch_A2_T3
00:46:40	4	A_2	A_1	A_2	kitch_A2_T4
00:50:00	0	A_3	A_1	A_3	kitch_A3_T0
00:53:20	2	A_0	A_2	A_3	kitch_A3_T2
00:56:40	3	A_0	A_0	A_3	kitch_A3_T3
01:00:00	4	A_3	A_0	A_3	kitch_A3_T4

G Neural network graphical representation

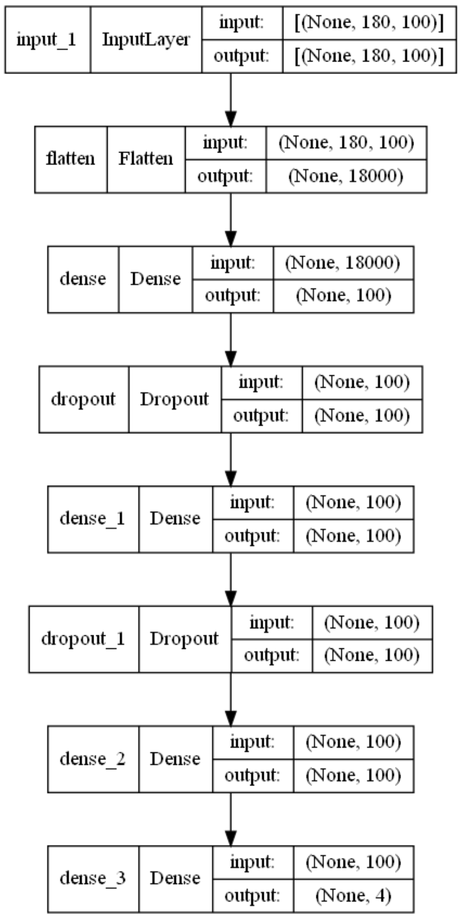

input_1 | InputLayer | input: [(None, 180, 100)] | output: [(None, 180, 100)]
flatten | Flatten | input: (None, 180, 100) | output: (None, 18000)
dense | Dense | input: (None, 18000) | output: (None, 100)
dropout | Dropout | input: (None, 100) | output: (None, 100)
dense_1 | Dense | input: (None, 100) | output: (None, 100)
dropout_1 | Dropout | input: (None, 100) | output: (None, 100)
dense_2 | Dense | input: (None, 100) | output: (None, 100)
dense_3 | Dense | input: (None, 100) | output: (None, 4)

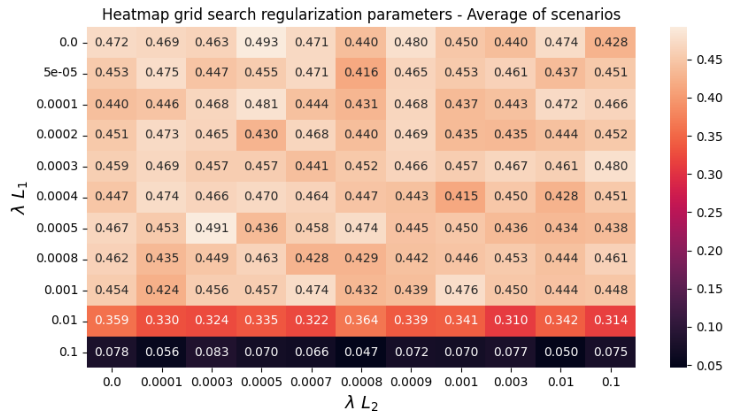
Figure G.1: Graphical representation of used neural network with 3 fully connected dense layers in the case of 4 output classes.

H Summarised overview datasets

Table H.1: Summarised overview of different sets and their purpose.

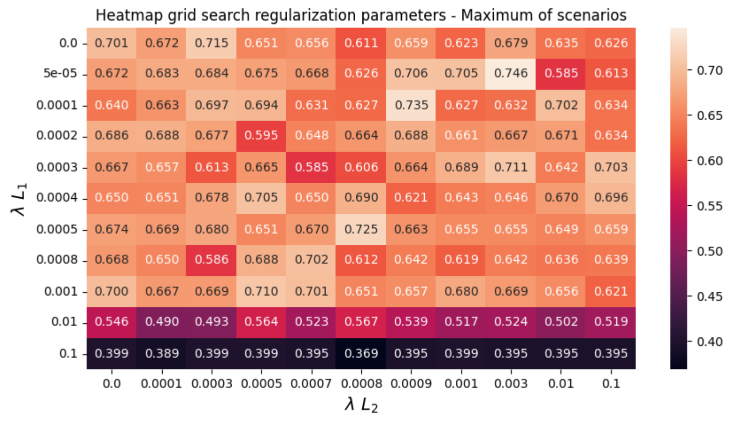
Dataset	**Data**	**Purpose**
Simple set	Separated per location for each receiving node	Perform initial activity recognition, confirming feasibility
Complete set	All activities and locations, separated for each receiving node	Benchmark performance, comparison to centralised ML, and comparison of aggregation algorithms
Location set	All activities and locations, in subsets for each activity location in which activity location L_i is either left out or the only node in the dataset	New locations
Client set	All activities and locations, in subsets for each client in which client n_i is either left out or the only node in the dataset	New clients

I Grid search supplementary results



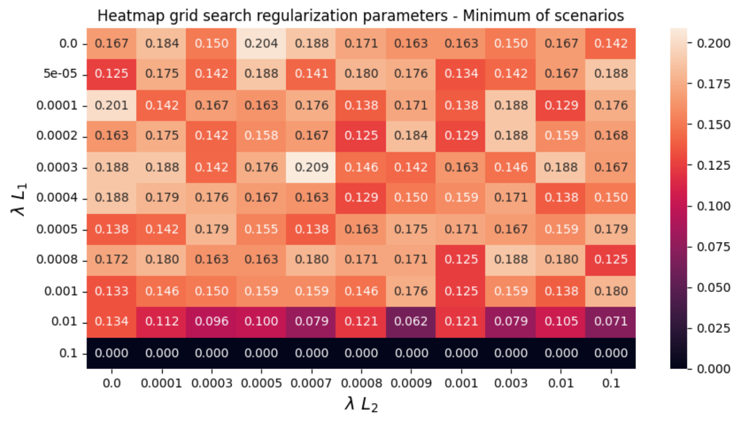
λL_1 \ λL_2	0.0	0.0001	0.0003	0.0005	0.0007	0.0008	0.0009	0.001	0.003	0.01	0.1
0.0	0.472	0.469	0.463	0.493	0.471	0.440	0.480	0.450	0.440	0.474	0.428
5e-05	0.453	0.475	0.447	0.455	0.471	0.416	0.465	0.453	0.461	0.437	0.451
0.0001	0.440	0.446	0.468	0.481	0.444	0.431	0.468	0.437	0.443	0.472	0.466
0.0002	0.451	0.473	0.465	0.430	0.468	0.440	0.469	0.435	0.435	0.444	0.452
0.0003	0.459	0.469	0.457	0.457	0.441	0.452	0.466	0.457	0.467	0.461	0.480
0.0004	0.447	0.474	0.466	0.470	0.464	0.447	0.443	0.415	0.450	0.428	0.451
0.0005	0.467	0.453	0.491	0.436	0.458	0.474	0.445	0.450	0.436	0.434	0.438
0.0008	0.462	0.435	0.449	0.463	0.428	0.429	0.442	0.446	0.453	0.444	0.461
0.001	0.454	0.424	0.456	0.457	0.474	0.432	0.439	0.476	0.450	0.444	0.448
0.01	0.359	0.330	0.324	0.335	0.322	0.364	0.339	0.341	0.310	0.342	0.314
0.1	0.078	0.056	0.083	0.070	0.066	0.047	0.072	0.070	0.077	0.050	0.075

(a)



λL_1 \ λL_2	0.0	0.0001	0.0003	0.0005	0.0007	0.0008	0.0009	0.001	0.003	0.01	0.1
0.0	0.701	0.672	0.715	0.651	0.656	0.611	0.659	0.623	0.679	0.635	0.626
5e-05	0.672	0.683	0.684	0.675	0.668	0.626	0.706	0.705	0.746	0.585	0.613
0.0001	0.640	0.663	0.697	0.694	0.631	0.627	0.735	0.627	0.632	0.702	0.634
0.0002	0.686	0.688	0.677	0.595	0.648	0.664	0.688	0.661	0.667	0.671	0.634
0.0003	0.667	0.657	0.613	0.665	0.585	0.606	0.664	0.689	0.711	0.642	0.703
0.0004	0.650	0.651	0.678	0.705	0.650	0.690	0.621	0.643	0.646	0.670	0.696
0.0005	0.674	0.669	0.680	0.651	0.670	0.725	0.663	0.655	0.655	0.649	0.659
0.0008	0.668	0.650	0.586	0.688	0.702	0.612	0.642	0.619	0.642	0.636	0.639
0.001	0.700	0.667	0.669	0.710	0.701	0.651	0.657	0.680	0.669	0.656	0.621
0.01	0.546	0.490	0.493	0.564	0.523	0.567	0.539	0.517	0.524	0.502	0.519
0.1	0.399	0.389	0.399	0.399	0.395	0.369	0.395	0.399	0.395	0.395	0.395

(b)



λL_1 \ λL_2	0.0	0.0001	0.0003	0.0005	0.0007	0.0008	0.0009	0.001	0.003	0.01	0.1
0.0	0.167	0.184	0.150	0.204	0.188	0.171	0.163	0.163	0.150	0.167	0.142
5e-05	0.125	0.175	0.142	0.188	0.141	0.180	0.176	0.134	0.142	0.167	0.188
0.0001	0.201	0.142	0.167	0.163	0.176	0.138	0.171	0.138	0.188	0.129	0.176
0.0002	0.163	0.175	0.142	0.158	0.167	0.125	0.184	0.129	0.188	0.159	0.168
0.0003	0.188	0.188	0.142	0.176	0.209	0.146	0.142	0.163	0.146	0.188	0.167
0.0004	0.188	0.179	0.176	0.167	0.163	0.129	0.150	0.159	0.171	0.138	0.150
0.0005	0.138	0.142	0.179	0.155	0.138	0.163	0.175	0.171	0.167	0.159	0.179
0.0008	0.172	0.180	0.163	0.163	0.180	0.171	0.171	0.125	0.188	0.180	0.125
0.001	0.133	0.146	0.150	0.159	0.159	0.146	0.176	0.125	0.159	0.138	0.180
0.01	0.134	0.112	0.096	0.100	0.079	0.121	0.062	0.121	0.079	0.105	0.071
0.1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

(c)

Figure I.1: Heat maps of grid search for regularisation parameter.

J Scenario combinations with and without regularisation

An extensive comparison is performed between all possible scenario combinations from the pilot study as training set. Certain training set combinations hold more valuable training information than others. Furthermore, to review generalisability to never seen data, each combination is evaluated on test data from each individual scenario. The training combinations that achieves the highest average F1-score over 7 folds are presented in Table J.1 for each evaluation data set. The scenarios are labeled as described in Table 4.2.

Table J.1: Best training set combinations for each evaluation data set, indicated by highest average F1-score over 7 folds, with variance of F1-scores over all training sets.

Setting	**Validation scenario**	**Best training scenarios (combinations possible)**	**Average F1-score**
No regularisation	1	[3, 4, 5]	0.4515 ± 0.0032
	2	2	0.7744 ± 0.0089
	3	[1, 3, 4]	0.5843 ± 0.0071
	4	4	0.8644 ± 0.0068
	5	5	0.6816 ± 0.0115
	6	6	0.8528 ± 0.0223
	7	7	0.6002 ± 0.0043
	8	8	0.7890 ± 0.0221
Regularisation	1	[2, 3, 6]	0.4825 ± 0.0040
	2	2	0.6797 ± 0.0078
	3	3	0.5938 ± 0.0079
	4	4	0.8035 ± 0.0074
	5	[2, 3, 5, 6]	0.6604 ± 0.0127
	6	6	0.8664 ± 0.0213
	7	7	0.6060 ± 0.0042
	8	8	0.8727 ± 0.0236

K Batch size impact supplementary results

Table K.1: Performance comparison batch sizes for L_{tv} (convergence time in seconds and F1-score).

Total epochs	**Rounds**	**Epochs**	**Conv (Batch 10)**	**Conv (Batch 20)**	**Conv (Batch 40)**	**F1 (Batch 10)**	**F1 (Batch 20)**	**F1 (Batch 40)**
500	50	10	3144.493	1812.182	628.393	0.734	0.731	0.764
	100	5	3309.9	2226.451	1039.169	0.758	0.75	0.773
	500	1	4068.903	2745.056	996.686	0.749	0.714	0.767
1000	100	10	2540.646	1867.306	567.427	0.926	0.925	0.931
	1000	1	3965.719	2852.682	1930.352	0.934	0.93	0.935
1250	250	5	3216.27	2233.854	1275.186	0.94	0.933	0.939
2500	250	10	2878.284	1444.766	908.44	0.921	0.922	0.931
	500	5	2932.404	1935.965	1182.25	0.939	0.942	0.936
5000	500	10	2924.742	1606.485	1029.816	0.922	0.927	0.93
	1000	5	3175.609	1870.491	1348.997	0.933	0.939	0.942
10000	1000	10	2701.522	1555.701	963.568	0.926	0.927	0.936

Table K.2: Performance comparison batch sizes for L_{table} (convergence time in seconds and F1-score).

Total epochs	**Rounds**	**Epochs**	**Conv (Batch 10)**	**Conv (Batch 20)**	**Conv (Batch 40)**	**F1 (Batch 10)**	**F1 (Batch 20)**	**F1 (Batch 40)**
500	50	10	3217.774	1715.242	1149.515	0.717	0.72	0.705
	100	5	3829.375	2114.815	1394.755	0.706	0.718	0.711
1000	100	10	3388.605	1798.383	1128.723	0.909	0.916	0.911
	1000	1	2624.954	2848.161	2375.351	0.932	0.935	0.929
1250	250	5	3380.855	2086.377	1446.371	0.926	0.926	0.924
2500	250	10	3295.268	1734.668	1085.054	0.907	0.908	0.912
5000	500	10	3811.313	1741.857	1147.939	0.906	0.913	0.907
	1000	5	4202.124	2251.675	1039.341	0.913	0.923	0.924
10000	1000	10	3004.064	2016.202	833.476	0.911	0.894	0.918

Table K.3: Performance comparison batch sizes for L_{kitch} (convergence time in seconds and F1-score).

Total epochs	**Rounds**	**Epochs**	**Conv (Batch 10)**	**Conv (Batch 20)**	**Conv (Batch 40)**	**F1 (Batch 10)**	**F1 (Batch 20)**	**F1 (Batch 40)**
500	50	10	3288.667	1712.633	998.971	0.702	0.694	0.665
	100	5	3628.862	2159.247	1206.22	0.712	0.716	0.665
1000	100	10	2990.846	1724.027	1027.069	0.89	0.914	0.934
	1000	1	2827.038	2677.56	1264.249	0.93	0.94	0.929
1250	250	5	3876.109	2104.037	879.184	0.914	0.931	0.927
2500	250	10	3177.511	1894.093	909.664	0.908	0.916	0.934
	500	5	2312.386	2128.365	895.885	0.919	0.924	0.922
5000	500	10	2038.572	1897.631	979.849	0.91	0.91	0.923
	1000	5	2658.381	2287.282	807.332	0.923	0.922	0.927
10000	1000	10	2795.752	1930.654	943.759	0.933	0.908	0.926

L Classical learning comparison supplementary results

The amount of epochs until convergence has been reached are indicated in Table L.1 for individual training settings. It is noteworthy to mention that the setting with 250 training rounds and a single epoch per round did not reach an accuracy of 0.8 in the federated learning setting.

Table L.1: Convergence speed comparison federated learning vs. classical machine learning.

Total epochs	**FL rounds**	**Local FL epochs**	**Epochs until 0.8 acc. (classical)**	**Epochs until 0.8 acc. (federated)**	**% Δ**
250	250	1	128	-	-
500	500	1	128	440	243.6
1250	250	5	130	350	169.2
2500	250	10	130	340	161.5

M Averaging algorithms supplementary results

Table M.1: Performance comparison averaging algorithms (convergence time in seconds).

Rounds	**Epochs**	**Conv (FedProx 1)**	**Conv (FedProx 0.1)**	**Conv (FedProx 0.01)**	**Conv (FedProx 0.001)**	**Conv (MimeLite, momentum)**	**Conv (MimeLite, no momentum)**	**Conv (FedAvg)**
250	1	-	-	-	-	-	-	-
	5	4393.694	4957.703	3490.36	7037.207	12834.88	12214.93	4024.418
	10	2972.59	4660.692	3119.44	7388.958	10495.74	10495.74	3027.094
500	1	4080.144	3773.323	4812.823	9571.518	13709.33	13709.33	5004.078
	5	4730.285	5349.892	7049.043	7418.643	12007.68	12007.68	1719.873

N Training with new locations supplementary results

Table N.1: Performance comparison unseen locations with different data fractions - L_{TV} (epochs until convergence and F1-score).

Total pre-trained epochs	**Pre-trained epochs**	**Pre-trained rounds**	**Conv (0.1)**	**Conv (0.2)**	**Conv (0.5)**	**Conv (0.75)**	**Conv (1)**	**F1 (0.1)**	**F1 (0.2)**	**F1 (0.5)**	**F1 (0.75)**	**F1 (1)**
500	5	100	585	345	170	105	85	0.638	0.668	0.703	0.697	0.709
	1	500	556	341	160	97	78	0.632	0.669	0.704	0.699	0.712
	10	50	610	320	170	110	80	0.649	0.662	0.710	0.707	0.714
1000	1	1000	267	167	70	48	34	0.695	0.743	0.762	0.765	0.763
	10	100	260	160	70	50	30	0.706	0.754	0.776	0.788	0.760
1250	5	250	250	160	70	45	35	0.711	0.758	0.775	0.774	0.772
2500	10	250	180	120	50	30	30	0.752	0.797	0.814	0.796	0.828
	5	500	225	180	65	45	35	0.698	0.754	0.761	0.767	0.770
5000	5	1000	300	175	80	55	40	0.705	0.750	0.771	0.775	0.772
	10	500	290	160	60	40	40	0.709	0.756	0.763	0.768	0.803

Table N.2: Performance comparison unseen locations with different data fractions - L_{table} (epochs until convergence and F1-score).

Total pre-trained epochs	**Pre-trained epochs**	**Pre-trained rounds**	**Conv (0.1)**	**Conv (0.2)**	**Conv (0.5)**	**Conv (0.75)**	**Conv (1)**	**F1 (0.1)**	**F1 (0.2)**	**F1 (0.5)**	**F1 (0.75)**	**F1 (1)**
500	5	100	680	475	210	135	100	0.642	0.693	0.727	0.725	0.719
	1	500	595	428	216	129	95	0.630	0.677	0.711	0.703	0.705
	10	50	680	440	200	140	100	0.649	0.693	0.716	0.726	0.726
1000	1	1000	302	196	101	64	42	0.694	0.743	0.756	0.761	0.759
	10	100	340	220	100	80	50	0.696	0.744	0.772	0.781	0.787
1250	5	250	300	190	80	60	40	0.693	0.734	0.753	0.765	0.757
2500	10	250	270	160	70	50	30	0.714	0.771	0.800	0.807	0.779
	5	500	375	225	110	75	50	0.712	0.744	0.773	0.782	0.774
5000	5	1000	350	240	105	70	55	0.698	0.739	0.772	0.759	0.775
	10	500	220	120	60	40	30	0.754	0.768	0.811	0.818	0.825

Table N.3: Performance comparison unseen locations with different data fractions - L_{kitch} (epochs until convergence and F1-score).

Total pre-trained epochs	**Pre-trained epochs**	**Pre-trained rounds**	**Conv (0.1)**	**Conv (0.2)**	**Conv (0.5)**	**Conv (0.75)**	**Conv (1)**	**F1 (0.1)**	**F1 (0.2)**	**F1 (0.5)**	**F1 (0.75)**	**F1 (1)**
500	5	100	630	345	170	115	95	0.629	0.672	0.708	0.716	0.723
	1	500	671	456	200	133	99	0.616	0.676	0.699	0.704	0.708
	10	50	720	400	210	140	100	0.634	0.670	0.721	0.722	0.720
1000	1	1000	268	176	80	54	37	0.668	0.710	0.754	0.753	0.748
	10	100	290	230	90	60	50	0.675	0.750	0.757	0.767	0.773
1250	5	250	295	210	95	65	45	0.686	0.745	0.758	0.766	0.766
2500	10	250	290	180	90	60	40	0.688	0.726	0.771	0.763	0.785
	5	500	260	195	80	60	40	0.677	0.732	0.751	0.763	0.756
5000	5	1000	325	215	105	65	45	0.683	0.733	0.768	0.761	0.760
	10	500	370	200	100	60	50	0.689	0.725	0.775	0.761	0.770

O Training with new clients supplementary results

O.1 New client - receiver only

Table O.1: Performance comparison new clients with different data fractions - receiver only (epochs until convergence and F1-score).

Added node	Totel pre-trained epochs	Conv (0.1)	Conv (0.2)	Conv (0.5)	Conv (0.75)	Conv (1)	F1 (0.1)	F1 (0.2)	F1 (0.5)	F1 (0.75)	F1 (1)
n_{kitch}	10	1160	662	295	199	154	0.587	0.635	0.685	0.692	0.705
	20	1118	606	267	181	139	0.568	0.635	0.677	0.683	0.693
	50	1224	646	289	194	145	0.594	0.640	0.687	0.685	0.687
	100	1048	571	241	163	124	0.612	0.651	0.675	0.686	0.691
	250	532	312	145	95	74	0.585	0.646	0.679	0.683	0.691
	500	147	100	47	33	26	0.625	0.667	0.710	0.720	0.720
	1000	114	59	31	22	17	0.702	0.733	0.768	0.783	0.784
	2000	98	54	25	18	14	0.719	0.767	0.807	0.806	0.814
n_{table}	10	1221	637	301	200	157	0.623	0.654	0.704	0.702	0.717
	20	1146	644	284	198	148	0.612	0.653	0.690	0.707	0.708
	50	1127	642	286	188	148	0.616	0.661	0.703	0.704	0.717
	100	1034	592	261	175	134	0.612	0.662	0.700	0.708	0.715
	250	493	286	145	101	75	0.614	0.658	0.707	0.714	0.713
	500	179	115	57	39	31	0.654	0.695	0.738	0.736	0.744
	1000	89	54	25	18	14	0.721	0.752	0.793	0.804	0.805
	2000	70	45	22	14	12	0.724	0.798	0.822	0.833	0.840
n_{ap}	10	1279	691	293	203	156	0.612	0.649	0.669	0.687	0.697
	20	1181	620	269	179	135	0.607	0.639	0.668	0.679	0.684
	50	1068	623	267	175	134	0.588	0.649	0.677	0.681	0.680
	100	1033	553	233	159	120	0.593	0.645	0.666	0.682	0.684
	250	419	227	106	66	53	0.607	0.647	0.674	0.669	0.684
	500	156	98	47	32	27	0.622	0.668	0.703	0.711	0.735
	1000	94	62	28	20	16	0.695	0.712	0.752	0.762	0.771
	2000	87	48	25	16	13	0.708	0.747	0.779	0.788	0.781
tv	10	1365	769	330	219	172	0.589	0.646	0.682	0.688	0.700
	20	1312	762	344	237	188	0.585	0.648	0.683	0.699	0.720
	50	1353	748	336	219	173	0.596	0.653	0.691	0.699	0.709
	100	1264	644	294	205	151	0.605	0.633	0.676	0.690	0.694
	250	605	361	178	119	89	0.575	0.637	0.694	0.687	0.693
	500	220	129	92	46	42	0.607	0.651	0.717	0.696	0.716
	1000	124	92	49	31	37	0.681	0.731	0.765	0.764	0.772
	2000	103	62	32	23	17	0.693	0.757	0.780	0.799	0.800

O.2 New client - transmitter only

Table O.2: Performance comparison new clients with different data fractions - transmitter only (epochs until convergence and F1-score).

Added node	**Pre-trained epochs**	**Pre-trained rounds**	**Conv (0.1)**	**Conv (0.2)**	**Conv (0.5)**	**Conv (0.75)**	**Conv (1)**	**F1 (0.1)**	**F1 (0.2)**	**F1 (0.5)**	**F1 (0.75)**	**F1 (1)**
n_{kitch}	1	10	-	1845	807	531	399	0.396	0.640	0.680	0.683	0.681
		20	-	1670	757	503	383	0.407	0.625	0.663	0.675	0.682
		50	-	1828	796	542	409	0.406	0.616	0.668	0.685	0.685
		100	-	1465	670	445	331	0.453	0.612	0.662	0.668	0.669
		250	1399	819	371	253	192	0.572	0.614	0.655	0.666	0.668
		500	626	355	168	109	86	0.631	0.668	0.708	0.712	0.723
		1000	383	240	105	70	49	0.685	0.744	0.763	0.769	0.756
		2000	373	204	107	67	49	0.722	0.767	0.803	0.803	0.806
	5	5	-	1925	845	530	415	0.384	0.639	0.682	0.665	0.679
		10	-	1710	760	500	385	0.433	0.619	0.658	0.663	0.675
		20	-	1775	745	530	385	0.440	0.647	0.670	0.706	0.690
		50	1405	895	405	280	205	0.569	0.624	0.662	0.681	0.675
		100	600	400	175	120	90	0.632	0.687	0.712	0.722	0.726
		250	340	260	110	70	60	0.693	0.755	0.772	0.774	0.787
		500	295	195	90	65	40	0.734	0.747	0.786	0.803	0.793
		1000	225	180	75	55	40	0.711	0.789	0.813	0.822	0.816
		2000	240	180	80	50	45	0.747	0.784	0.806	0.802	0.818
	10	5	-	1900	790	530	410	0.397	0.645	0.677	0.691	0.696
		10	-	1620	700	470	350	0.440	0.637	0.679	0.681	0.685
		20	1730	1070	450	300	240	0.584	0.647	0.671	0.678	0.700
		50	640	440	190	140	100	0.616	0.666	0.698	0.720	0.712
		100	430	240	110	70	60	0.711	0.756	0.768	0.773	0.788
		250	350	190	80	60	40	0.749	0.787	0.800	0.818	0.806
		500	280	140	80	50	40	0.751	0.755	0.813	0.805	0.815
n_{table}	1	5	-	1808	741	484	361	0.403	0.636	0.659	0.669	0.665
		10	-	1891	774	514	389	0.407	0.635	0.663	0.670	0.671
		20	-	1918	817	541	399	0.395	0.627	0.663	0.667	0.655
		50	-	1779	774	523	393	0.422	0.633	0.667	0.681	0.677
		100	-	1618	641	422	313	0.471	0.634	0.652	0.653	0.650
		250	1180	720	297	208	158	0.570	0.627	0.652	0.664	0.668
		500	453	275	125	82	61	0.644	0.675	0.706	0.697	0.706
		1000	243	139	65	46	34	0.695	0.733	0.760	0.765	0.764
		2000	229	133	47	32	25	0.719	0.772	0.789	0.796	0.784
	5	5	-	1910	770	515	390	0.396	0.632	0.651	0.669	0.674
		10	-	1795	780	515	380	0.403	0.631	0.665	0.668	0.665
		20	-	1565	640	435	310	0.477	0.638	0.654	0.680	0.660
		50	1310	710	315	205	155	0.593	0.619	0.654	0.653	0.658
		100	515	290	115	75	60	0.666	0.682	0.703	0.698	0.710
		250	180	120	65	40	30	0.705	0.747	0.780	0.787	0.784
		500	200	135	60	35	30	0.684	0.784	0.801	0.790	0.793
		1000	185	110	55	30	25	0.735	0.760	0.806	0.792	0.795
		2000	195	125	55	35	25	0.721	0.794	0.797	0.805	0.790
	10	5	-	1920	780	510	380	0.396	0.652	0.658	0.671	0.667
		10	-	1500	620	410	300	0.486	0.642	0.662	0.667	0.656
		20	1830	950	390	260	210	0.591	0.632	0.650	0.656	0.680
		50	460	310	110	80	60	0.630	0.699	0.688	0.705	0.704
		100	310	160	70	40	30	0.705	0.758	0.775	0.767	0.766
		250	210	120	50	40	30	0.745	0.790	0.811	0.837	0.835
		500	230	140	60	40	30	0.729	0.779	0.808	0.811	0.792
n_{ap}	1	5	-	-	1221	798	624	0.348	0.486	0.674	0.681	0.694
		10	-	-	1282	867	669	0.343	0.491	0.663	0.676	0.692
		20	-	-	1102	743	568	0.371	0.549	0.672	0.673	0.686
		50	-	-	1064	728	550	0.378	0.556	0.659	0.680	0.685
		100	-	-	996	663	520	0.384	0.594	0.661	0.668	0.679
		250	1575	963	467	323	245	0.568	0.623	0.666	0.678	0.677
		500	655	481	260	186	134	0.588	0.651	0.689	0.706	0.699
		1000	411	262	120	86	71	0.637	0.696	0.716	0.730	0.735
		2000	330	247	108	85	67	0.657	0.714	0.751	0.753	0.763
	5	5	-	-	1215	845	630	0.343	0.501	0.676	0.697	0.691
		10	-	-	1100	745	575	0.365	0.535	0.671	0.681	0.693
		20	-	-	970	640	515	0.402	0.592	0.677	0.669	0.697
		50	-	-	605	410	310	0.570	0.645	0.677	0.689	0.688
		100	985	565	305	200	155	0.620	0.646	0.696	0.703	0.704
		250	505	305	130	95	75	0.658	0.716	0.733	0.744	0.751
		500	360	250	115	75	65	0.671	0.728	0.761	0.764	0.770
		1000	255	195	105	70	55	0.668	0.729	0.767	0.773	0.779
		2000	250	170	100	65	55	0.691	0.730	0.777	0.765	0.780
	10	5	-	-	1100	740	550	0.400	0.570	0.688	0.690	0.693
		10	-	-	1020	680	530	0.407	0.579	0.669	0.673	0.683
		20	-	-	690	450	330	0.544	0.635	0.688	0.692	0.684
		50	910	470	240	180	130	0.617	0.648	0.694	0.712	0.708
		100	420	250	150	100	70	0.644	0.683	0.737	0.747	0.736
		250	310	220	100	70	60	0.668	0.742	0.774	0.777	0.790
		500	270	200	130	70	60	0.674	0.730	0.773	0.768	0.783
n_{tv}	1	5	-	-	694	470	357	0.468	0.628	0.661	0.678	0.680
		10	-	-	704	464	341	0.464	0.625	0.653	0.662	0.657
		20	-	-	737	486	374	0.461	0.658	0.681	0.685	0.692
		50	-	-	712	482	366	0.477	0.637	0.669	0.681	0.687
		100	-	-	596	407	311	0.528	0.634	0.656	0.670	0.674
		250	1211	725	341	241	178	0.580	0.613	0.662	0.680	0.677
		500	458	302	140	91	68	0.628	0.673	0.707	0.711	0.710
		1000	279	166	80	52	41	0.715	0.745	0.775	0.775	0.777
		2000	216	130	60	39	32	0.734	0.784	0.800	0.797	0.806
	5	5	-	-	690	460	355	0.461	0.623	0.658	0.674	0.682
		10	-	-	700	455	345	0.480	0.656	0.685	0.679	0.683
		20	-	-	600	400	305	0.540	0.636	0.672	0.677	0.687
		50	1375	715	330	220	165	0.619	0.634	0.671	0.678	0.677
		100	380	235	125	85	65	0.665	0.692	0.738	0.739	0.748
		250	285	150	65	45	35	0.752	0.761	0.799	0.799	0.804
		500	250	110	65	45	30	0.760	0.776	0.819	0.824	0.812
		1000	185	110	60	40	30	0.760	0.781	0.818	0.812	0.810
		2000	155	95	45	30	25	0.733	0.759	0.817	0.812	0.826
	10	5	-	-	690	470	340	0.476	0.647	0.690	0.703	0.688
		10	-	-	720	450	350	0.483	0.641	0.683	0.667	0.677
		20	1790	1050	460	310	250	0.586	0.634	0.657	0.667	0.692
		50	500	330	140	100	70	0.676	0.722	0.736	0.755	0.748
		100	240	140	70	50	40	0.711	0.754	0.791	0.798	0.802
		250	190	130	50	40	30	0.764	0.785	0.807	0.831	0.825
		500	180	130	60	40	30	0.764	0.805	0.836	0.844	0.826

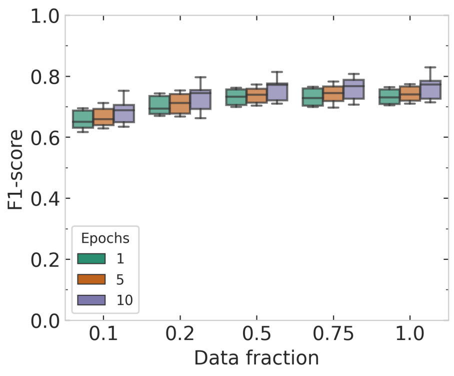
O.3 New client - complete network

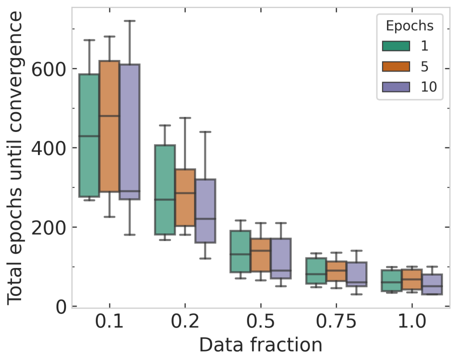
Table O.3: Performance comparison new clients with different data fractions - complete network (epochs until convergence and F1-score).

Added node	**Pre-trained epochs**	**Pre-trained rounds**	**Conv (0.1)**	**Conv (0.2)**	**Conv (0.5)**	**Conv (0.75)**	**Conv (1)**	**F1 (0.1)**	**F1 (0.2)**	**F1 (0.5)**	**F1 (0.75)**	**F1 (1)**
n_{kitch}	1	10	-	-	1281	849	645	0.339	0.511	0.700	0.702	0.706
		20	-	-	1132	754	578	0.349	0.538	0.677	0.683	0.689
		50	-	-	1235	813	611	0.329	0.494	0.683	0.691	0.696
		100	-	-	1049	699	544	0.363	0.561	0.691	0.695	0.708
		250	-	1339	669	431	337	0.533	0.620	0.685	0.686	0.697
		500	1021	590	314	210	152	0.605	0.658	0.714	0.722	0.720
		1000	635	353	187	118	92	0.667	0.695	0.749	0.749	0.758
		2000	658	355	169	125	92	0.689	0.745	0.782	0.791	0.785
	5	5	-	-	1240	830	635	0.322	0.501	0.689	0.695	0.710
		10	-	-	1195	800	610	0.349	0.523	0.683	0.699	0.698
		20	-	-	1205	805	625	0.348	0.506	0.698	0.706	0.722
		50	-	1630	705	465	365	0.513	0.664	0.685	0.691	0.709
		100	1265	725	345	230	170	0.633	0.682	0.718	0.723	0.723
		250	590	435	180	125	85	0.675	0.739	0.761	0.762	0.755
		500	425	335	145	95	75	0.702	0.737	0.770	0.774	0.783
		1000	560	290	130	85	70	0.744	0.767	0.776	0.788	0.795
		2000	475	310	150	95	70	0.712	0.752	0.789	0.791	0.785
	10	5	-	-	1220	800	620	0.328	0.505	0.698	0.694	0.708
		10	-	-	1150	780	580	0.374	0.558	0.706	0.712	0.715
		20	-	1750	830	530	410	0.490	0.656	0.711	0.706	0.718
		50	1460	730	370	270	200	0.631	0.660	0.707	0.730	0.727
		100	640	480	190	120	90	0.674	0.746	0.761	0.751	0.754
		250	590	260	150	100	70	0.726	0.744	0.790	0.794	0.783
		500	450	330	150	100	70	0.704	0.769	0.800	0.803	0.789
n_{table}	1	5	-	-	1148	767	585	0.353	0.543	0.684	0.692	0.699
		10	-	-	1192	824	620	0.350	0.524	0.675	0.691	0.695
		20	-	-	1216	844	637	0.330	0.486	0.673	0.693	0.690
		50	-	-	1215	816	621	0.357	0.518	0.682	0.695	0.699
		100	-	-	1078	681	524	0.369	0.558	0.684	0.679	0.691
		250	-	1409	630	406	311	0.558	0.653	0.685	0.686	0.694
		500	890	581	294	200	150	0.629	0.668	0.712	0.716	0.716
		1000	456	328	166	99	76	0.666	0.729	0.756	0.761	0.759
		2000	353	221	118	62	61	0.707	0.757	0.778	0.780	0.788
	5	5	-	-	1245	835	635	0.327	0.498	0.684	0.699	0.700
		10	-	-	1210	815	605	0.336	0.488	0.680	0.693	0.692
		20	-	-	1085	720	540	0.380	0.566	0.690	0.696	0.697
		50	-	1485	615	405	310	0.569	0.674	0.683	0.682	0.690
		100	1015	480	280	180	140	0.653	0.665	0.713	0.718	0.724
		250	425	300	135	80	65	0.706	0.753	0.775	0.768	0.777
		500	365	240	100	70	50	0.698	0.760	0.780	0.796	0.781
		1000	410	245	110	65	50	0.754	0.767	0.806	0.797	0.795
		2000	385	210	120	85	55	0.735	0.761	0.797	0.788	0.797
	10	5	-	-	1180	810	630	0.346	0.515	0.680	0.693	0.712
		10	-	-	1090	710	530	0.384	0.587	0.699	0.704	0.704
		20	-	1790	770	490	370	0.516	0.675	0.700	0.694	0.694
		50	1020	570	270	200	150	0.637	0.678	0.701	0.726	0.726
		100	620	310	140	100	70	0.713	0.736	0.765	0.770	0.763
		250	390	200	100	70	50	0.739	0.767	0.788	0.798	0.785
		500	470	290	120	80	60	0.738	0.771	0.791	0.802	0.799
n_{ap}	1	5	-	-	1786	1198	897	0.307	0.399	0.689	0.701	0.698
		10	-	-	1832	1244	946	0.305	0.408	0.684	0.695	0.698
		20	-	-	1520	1085	818	0.321	0.433	0.660	0.688	0.698
		50	-	-	1595	1096	836	0.341	0.439	0.682	0.693	0.705
		100	-	-	1484	1027	773	0.344	0.468	0.668	0.689	0.690
		250	-	1758	796	540	422	0.517	0.638	0.677	0.687	0.697
		500	1378	871	448	322	245	0.580	0.632	0.686	0.700	0.702
		1000	763	549	265	193	153	0.633	0.674	0.719	0.738	0.734
		2000	677	395	221	156	131	0.650	0.698	0.732	0.748	0.752
	5	5	-	-	1780	1190	910	0.303	0.407	0.694	0.702	0.710
		10	-	-	1650	1110	855	0.322	0.445	0.689	0.696	0.710
		20	-	-	1440	980	750	0.341	0.476	0.685	0.692	0.702
		50	-	-	965	645	510	0.460	0.630	0.689	0.696	0.708
		100	1620	1085	515	380	305	0.604	0.660	0.693	0.709	0.728
		250	660	550	275	200	150	0.629	0.686	0.733	0.745	0.749
		500	665	505	235	165	120	0.672	0.720	0.747	0.752	0.753
		1000	530	445	205	135	110	0.666	0.733	0.750	0.754	0.753
		2000	525	415	205	135	110	0.671	0.728	0.754	0.769	0.764
	10	5	-	-	1510	1070	790	0.347	0.445	0.678	0.703	0.700
		10	-	-	1500	1020	780	0.349	0.461	0.679	0.690	0.701
		20	-	-	1020	720	540	0.455	0.607	0.690	0.702	0.695
		50	1660	980	490	350	240	0.630	0.667	0.712	0.726	0.715
		100	830	710	310	230	160	0.642	0.715	0.733	0.756	0.744
		250	550	390	210	150	120	0.667	0.728	0.760	0.776	0.779
		500	610	370	230	160	130	0.679	0.724	0.763	0.779	0.777
n_{tv}	1	5	-	-	1124	755	577	0.374	0.545	0.683	0.695	0.702
		10	-	-	1194	804	586	0.364	0.548	0.682	0.701	0.695
		20	-	-	1234	852	645	0.373	0.537	0.695	0.703	0.706
		50	-	-	1207	787	628	0.375	0.534	0.686	0.693	0.704
		100	-	-	1102	727	554	0.397	0.575	0.690	0.696	0.703
		250	-	1475	686	481	369	0.544	0.639	0.683	0.699	0.709
		500	976	627	327	240	176	0.610	0.654	0.702	0.715	0.719
		1000	610	374	204	140	104	0.680	0.725	0.743	0.764	0.754
		2000	423	277	127	93	72	0.690	0.738	0.761	0.772	0.774
	5	5	-	-	1235	825	625	0.355	0.524	0.697	0.700	0.705
		10	-	-	1170	820	610	0.388	0.560	0.699	0.716	0.717
		20	-	-	1105	750	550	0.422	0.592	0.702	0.716	0.712
		50	-	1585	670	455	340	0.576	0.682	0.697	0.713	0.709
		100	880	680	290	225	160	0.643	0.708	0.719	0.739	0.736
		250	525	395	185	125	90	0.688	0.744	0.769	0.778	0.775
		500	415	365	135	100	85	0.727	0.766	0.773	0.784	0.790
		1000	420	305	140	95	80	0.734	0.773	0.781	0.787	0.798
		2000	385	270	120	85	60	0.712	0.771	0.793	0.797	0.795
	10	5	-	-	1140	770	600	0.385	0.577	0.700	0.707	0.726
		10	-	-	1220	830	610	0.383	0.539	0.702	0.709	0.706
		20	-	1820	910	610	460	0.491	0.640	0.700	0.712	0.710
		50	980	790	370	220	160	0.644	0.711	0.741	0.739	0.737
		100	640	380	180	130	100	0.706	0.743	0.770	0.782	0.783
		250	440	280	130	100	70	0.726	0.753	0.778	0.794	0.789
		500	520	250	140	80	60	0.727	0.761	0.803	0.802	0.799

P Limited data availability supplementary results

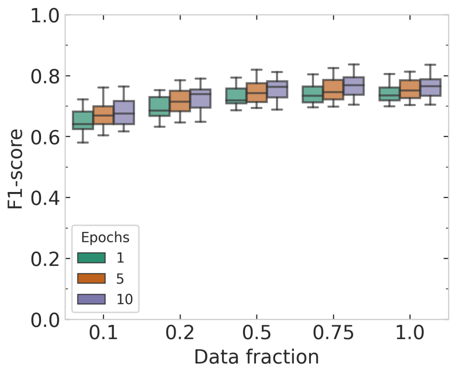
The effect of different amounts of local computation (1, 5 or 10 epochs per round) is depicted below, for both the *client* and *location* set.

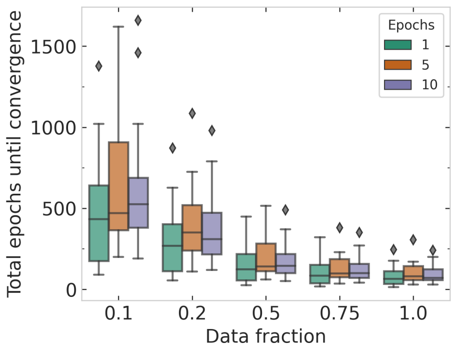




(a) (b)

Figure P.1: Impact of amount of data available during optimisation on pre-existing models (*location* set) based on different amount of local computation, with a) the impact on F1-score, b) the impact on the amount of total training epochs until convergence is reached.





(a) (b)

Figure P.2: Impact of amount of data available during optimisation on pre-existing models (*client* set) based on different amount of local computation, with a) the impact on F1-score, b) the impact on the amount of total training epochs until convergence is reached.