TÉCNICO LISBOA

Federated Learning for Predicting the Next Node in Action Flows

Daniel Francisco Lopes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Luís Eduardo Teixeira Rodrigues
João Pedro Nunes Nadkarni

Examination Committee

Chairperson: Prof. Valentina Nisi
Supervisor: Prof. Luís Eduardo Teixeira Rodrigues
Member of the Committee: Prof. Manuel Fernando Cabido Peres Lopes

November 2022

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

Firstly, I would like to thank my parents and my brother for all the support and encouragement they have given me over the years and, especially, over the past year. You have experienced firsthand how hard it was and have been there for me through thick and thin. Thank you for always believing in me and my capabilities and for helping me overcome the most challenging year I have faced in my life. Without your support, I am sure I would not have been able to focus on myself and the work I had to do. Also, thank you to my grandparents, uncles, aunts and cousins for their support. To my family, thank you for helping me become who I strive to be.

Secondly, I would like to thank Professor Luís Rodrigues along with João Nadkarni, Filipe Assunção and Miguel Lopes from OUTSYSTEMS for initially contacting me to work on this thesis and for taking me under their supervision. Also, thank you, for the several meetings, discussions, suggestions and feedback throughout the development of this thesis. This thesis was only possible due to your experience, dedication, knowledge, and insights. A special thank you to OUTSYSTEMS for helping me with anything I have needed, particularly to the Artificial Intelligence Development Experiences (AIDE) team for welcoming me with open arms.

Lastly, to my friends Margarida, Rodrigo, José, Alexandre, Mariana and Marta, thank you for your support over this difficult period and for being there for me even if, for the past few years, I was not as present as I should have been.

To each and every one of you – Thank you.

This work was done in the scope of a curricular internship at OUTSYSTEMS and was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) as part of the project with reference UIDB/50021/2020.

Abstract

Federated learning is a machine learning approach that allows different clients to collaboratively train a common model without sharing their data sets. We focus on centralized federated learning, where a central server collects contributions from individual clients, merges these contributions, and disseminates the results to all clients. Since clients have different data and classify data differently, there is a trade-off between the generality of the common model and the personalization of the classification results. Current approaches rely on using a combination of a global model, common to all clients, and multiple local models, that support personalization. In this work, we report the results of a study, where we have applied some of these approaches to a concrete use case, namely the *Service Studio* platform from OUTSYSTEMS, where Graph Neural Networks help programmers in the development of applications. Furthermore, we explore two different approaches which merge some of the state-of-the-art algorithms so as to develop the best model for all the different clients. Our results show that one of the proposed approaches manages to achieve similar performance to the best-performing algorithms for all the classes of clients and can even outperform previous algorithms for some classes of clients.

Keywords

Federated Learning; Personalized Federated Learning; Graph Neural Networks.

Resumo

A aprendizagem federada é uma abordagem de aprendizagem automática que permite que diversos clientes treinem um único modelo colaborativamente sem necessitarem de partilhar os seus dados. O nosso foco é na aprendizagem federada centralizada, onde um servidor centralizado coleciona as contribuições individuais dos clientes, agrega-as, e dissemina os resultados por todos os clientes. Uma vez que os clientes mantêm e classificam dados diferentes, existe um compromisso entre a generalidade do modelo obtido e a personalização das classificações. Na literatura, é possível encontrar soluções para este problema que se baseiam na divisão do modelo em duas partes: uma parte global, comum a todos os clientes, e uma parte local especializada a cada cliente, que suporta a personalização. Neste trabalho realizamos um estudo sobre o desempenho de algumas destas abordagens quando aplicadas num produto da OUTSYSTEMS, onde redes neuronais para grafos são usadas para ajudar os programadores no desenvolvimento de aplicações. Para além disso, exploramos duas abordagens diferentes que combinam alguns dos algoritmos mais recentes de forma a desenvolver o melhor modelo para todos os clientes. Dos resultados obtidos, verifica-se que uma das abordagens propostas consegue atingir um desempenho semelhante aos algoritmos com melhor desempenho para todas as classes de clientes, conseguindo mesmo superar os algoritmos anteriores para algumas classes de clientes.

Palavras Chave

Aprendizagem Federada; Aprendizagem Federada Personalizada; Redes Neuronais para Grafos.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AI	Artificial Intelligence
CMFL	Communication-Mitigated Federated Learning
DP	Differential Privacy
FATE	Federated AI Technology Enabler
FedAvg	Federated Averaging
FedBABU	Federated Averaging with Body Aggregation and Body Update
FedPer	Federated Learning with Personalization Layers
FedRep	Federated Representation Learning
FL	Federated Learning
GAN	Generative Adversarial Network
GNN	Graph Neural Network
LG-FedAvg	Local Global Federated Averaging
MAML	Model Agnostic Meta Learning
mGAN-AI	multi-task GAN for Auxiliary Identification
ML	Machine Learning
MLP	Multilayer Perceptron
Per-FedAvg	Personalized Federated Averaging
SGD	Stochastic Gradient Descent
SMC	Secure Multiparty Computation

1

Introduction

Contents

1.1 Motivation

Machine Learning (ML) is an area of Artificial Intelligence (AI) that studies how to build a model, from a given training data set, such that it can be used to predict an output given an input. Federated Learning (FL) is a particular case of ML where different entities collaborate to construct a common model without explicitly exchanging their data sets and compromising performance while, ideally, preserving the privacy of their data. Our research is driven by the requirements of OUTSYSTEMS, where FL is being explored as an alternative to the current fully centralized inference and training setup, in order to build a model intended to help programmers in their coding tasks.

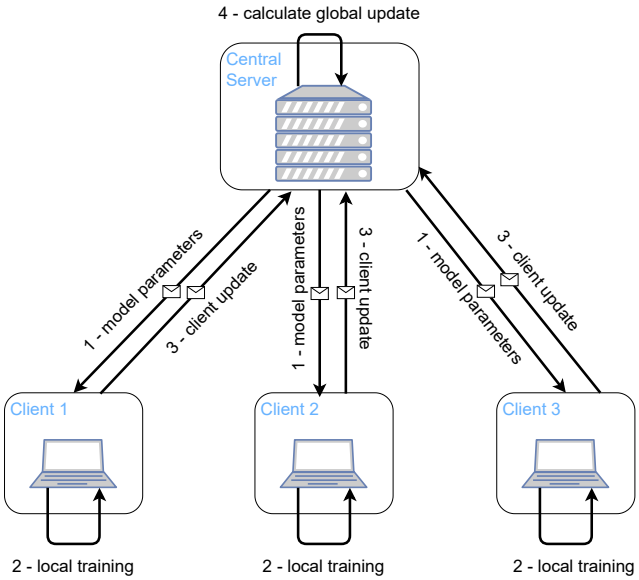


Figure 1.1: Example of the steps of a single FL communication round

In our work, we study the centralized FL approach, which uses a central server to keep a global model. The server periodically performs communication rounds with some clients (all or just a subset), to improve the global model with the help of the individual training data from each client. Figure 1.1 illustrates the procedure for a single communication round with three clients. In each communication round, the selected clients receive the global model parameters from the central server (step 1), train this model with their private data (step 2), and send back to the server the resulting updates to the model (step 3). The server then aggregates all the received local updates to generate a global update (step 4) to improve the global model. This procedure is repeated over various communication rounds.

FL has many challenges. First, the communication rounds may consume significant processing and network resources and should be made as efficient as possible. Second, keeping data at the clients may not be enough to preserve privacy, as it may be possible to infer the content of the training data from the updates to the model. Third, a faulty or malicious client may attempt to bias or poison the global model.

Lastly, clients may have different data and different classification preferences, which creates the need for maintaining personalized models, in combination with a common general model.

In this work, we are mainly concerned with the last challenge, particularly, in techniques that can offer clients personalized models, while still benefiting from FL. Current approaches for personalization rely on using a global model, common to all clients, which is then adapted to each client's data, or on splitting the model in two and having a shared global part and multiple more specific parts, each one tailored and maintained exclusively by each client. We survey the state-of-the-art solutions for FL and identify unexplored alternatives for training in this personalized setup that are worth exploring. Based on these findings, we propose to implement and evaluate some of the existent solutions and our two new variants, *FedHybridAvgLG* and *FedHybridAvgLGDual*.

We experiment and evaluate these new variants in the context of the *Service Studio* platform from OUTSYSTEMS. *Service Studio* is a low-code platform that allows users to design and manage systems and applications in a simple and efficient manner through a visual and interactive user interface. In this platform, among other things, the user defines the application logic by creating a flow of actions. These actions can be of several types, for instance, "`if`", "`for`" or "`assign`" (many other actions related to, for example, user interface development and data management, are possible). In this platform, ML is used to give recommendations to the users about which actions should be added next to an action flow.

An action flow can be modelled as a graph where the actions are nodes and the edges represent the flow from action to action. The graph can then be used as input to a specific type of ML neural network model architecture, a Graph Neural Network (GNN), which is specialized in interpreting graphs and making predictions on them. In our case, the model predicts, from a finite set of possible node types, which are the most probable to be added next to the graph. This prediction is then used by *Service Studio* to recommend possible next actions to the user. The use of FL in this context is relevant because it allows the model to be trained using contributions from various clients while ensuring that information about the applications being developed remains private.

1.2 Contributions

This thesis surveys some of the existing approaches for model personalization in FL that have been proposed in the literature in order to analyse and identify possible new alternatives that are fit for the use case of OUTSYSTEMS. Hence, the thesis offers the following contributions:

- It presents a survey of the key challenges of FL and of the main approaches to address these challenges. The covered challenges are communication efficiency, privacy and security concerns and model personalization.
- It proposes two new approaches, *FedHybridAvgLG* and *FedHybridAvgLGDual*, that combine pre-

vious algorithms and can automatically select the algorithm to be used for each client, based on the number of data points in the client's data set.

1.3 Results

This thesis has produced the following results:

- An implementation of several algorithms for personalized FL in a common framework, namely the Flower [2] framework, to support their comparison, which is available in the following GitHub repository: github.com/OS-danielfranciscolopes/FL-Personalization.
- An experimental study of the performance of all the implemented algorithms in the use case of OUTSYSTEMS. We aim at understanding what are the best approaches to be used in our use case. The results show that one of the new approaches proposed in this thesis achieves a performance similar to the top-performing algorithms for each class of clients.

1.4 Research History

This work was performed in the context of a curricular internship with OUTSYSTEMS. As noted above, the recommendation system that OUTSYSTEMS uses in production has no support for personalization. The work described in this thesis is a step towards the goal of supporting personalized models with the aim of improving the accuracy of the models.

Parts of the work described in this thesis have been published in a national conference and as a poster in a workshop of NeurIPS:

- D. Lopes, J. Nadkarni, F. Assunção, M. Lopes and L. Rodrigues. "Aprendizagem Federada para Previsão do Próximo Nó em Fluxos de Ações", in *Actas do décimo terceiro Simpósio de Informática (Inforum)*, Guarda, Portugal, September 2022.
- D. Lopes, J. Nadkarni, F. Assunção, M. Lopes and L. Rodrigues, "Poster: Federated Learning for predicting the next node in action flows", in *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*, New Orleans, LA, USA, December 2022. [Online]. Available: https://openreview.net/forum?id=lx59l_Aq12r.

1.5 Organization of the Document

This thesis is organized as follows: Chapter 1 provides an introduction to the thesis; Chapter 2 presents all the background related to our work; Chapter 3 references some of the state-of-art work for the sev-

eral challenges related to FL; Chapter 4 introduces the context of OUTSYTEMS and describes the proposed algorithms as well as their implementation and of some personalization algorithms from the literature; Chapter 5 presents the results of an experimental study performed to analyse which are the best-performing algorithms for the OUTSYSTEMS use case; finally, Chapter 6 concludes the thesis and presents some possible directions for future work.

2

Background

Contents

In this chapter, we present the background relevant to our work. We start by a simple introduction to ML (Section 2.1) and one of the types of ML models, the GNNs (Section 2.2), then we present FL as an ML approach (Section 2.3). We then go over some of the challenges of FL, namely, communication efficiency (Section 2.4), client data privacy (Section 2.5), model security (Section 2.6) and model personalization (Section 2.7). Finally, we explore some of the work done for GNNs in FL (Section 2.8) and some of the existing FL frameworks (Section 2.9).

2.1 Machine Learning

ML is one of the branches of the vast area of AI. ML leverages algorithms that receive data as input, known as training data, to build a model based on that data. Given a data point as input, these models can make predictions or decisions on a specific task for which they were trained on. If the training data is carefully selected, the quality of the model can improve as more training data is provided to the model. For this reason, one often states that ML models can learn from experience. It should be noted, however, that if the training data does not capture the diversity of the expected inputs, the resulting model may be inadequate; for instance, a model can learn details or noise from the training data that may lead to poor performance (a problem known as overfitting).

The task of building an ML model usually uses three phases: training, validation, and testing. The data set is partitioned into three disjoint partitions, one for each stage. This division allows for assessing the generalization capabilities of the model by evaluating it with data that was not used to build the model. In this work, we focus on the *supervised* training setting, where for each training data point the expected output is also provided (other settings, such as *semi-supervised* and *unsupervised* training are out of the scope of this study). The ML algorithm uses the training data to find data patterns which allow it to derive the expected output from the received input. The validation step, where the model performance is evaluated, is normally performed at the end of each training epoch (a single pass of the training partition) and can be used to tune hyperparameters, such as the learning rate, and to avoid overfitting to the training data (overfitting can be detected if the performance in validation data is decreasing while the performance in training data keeps increasing). Finally, after the training and validation of the model are completed, we have the testing phase. In this phase, the model receives new unseen data from the test partition and its performance is evaluated by comparing the outputs produced with the expected output, also known as the ground truth. Unlike the validation data, the testing data should not be used to fine-tune the model, nor to avoid overfitting during the training stage.

There are several types and architectures of ML models. In this work, we focus on Artificial Neural Networks and, particularly, on GNNs [1], as they are the architecture used by the AI-powered capabilities present in *Service Studio*. GNNs learn from data sets that are modelled as graphs. Other neural

network architectures are better suited for other types of data, such as Convolutional Neural Networks, suitable for data presented as a grid (such as images), or Recurrent Neural Networks, for sequential data. GNNs itself can have different architectures, the one used by OUTSYSTEMS is the Message Passing architecture proposed by Battaglia et al. [1].

2.2 Graph Neural Networks

GNNs are a type of ML model which can interpret and make predictions on data that is structured as a graph. We consider graphs $G = (u, V, E)$ composed of a set of nodes V, a set of edges E, and a set of global attributes u. Each node has its own attributes, where v_i represents the attributes for node i. Edges also have attributes, where e_{ij} represents the attributes of the edge with node i as source and node j as destination. Lastly, graphs can also have global attributes, represented by u; these are attributes common to the whole graph. Using the example given by Battaglia et al. [1], if we consider a gravitational field where a set of balls are connected between each other by springs (the connections are arbitrary, so one ball might not be connected to all the other balls), then we can easily model a graph where the balls are the nodes and the springs connecting the balls are the edges. In this example, one of the node attributes of v_i might be the mass of the ball, one of the edge attributes of e_{ij} might be the spring constant and one of the global attributes of u might be the gravitational field.

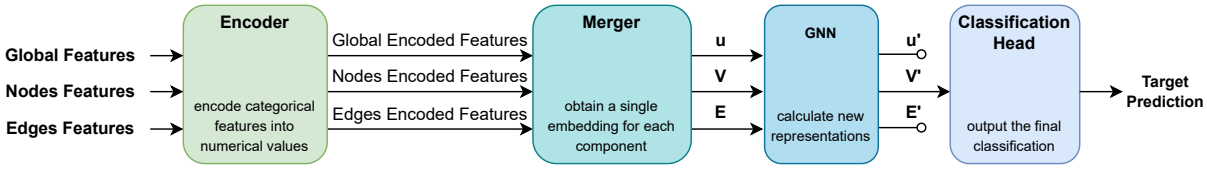


Figure 2.1: Pipeline of the GNN model used

Figure 2.1 illustrates the pipeline of the GNN used in this work, which is composed of the Encoder, the Merger, the GNN model and the Classification Head. As seen in the figure, before passing the graph to the model itself, some graph transformations are performed by the Encoder and the Merger. That is because each graph component (node, edge, global), represented by its attributes, needs to be mapped to a representation that can be interpreted by the GNN, namely, each component needs to be represented as a vector in a multidimensional space. This process occurs in two phases: i) a phase where each categorical attribute is mapped to a numerical representation, completed after the Encoder and; ii) a phase where each graph component (now composed of numerical attributes) is mapped into a multidimensional space that captures the components' "proximity", performed by the Merger.

2.2.1 Encoder

Some of the attributes of the inputs might be categorical and thus need to be transformed into numerical values which can be interpreted. Going back to the example of the first paragraph of this section, if we had the colour of each ball as a node attribute, that attribute would be categorical, so, it needs to be mapped to some numerical representation. Particularly, in our work, all of the attributes are categorical. Therefore, before passing the graph to the network there needs to be a process of feature encoding where each categorical attribute or feature is transformed into numerical values. There are several techniques to transform categorical values into numerical values, though we will only discuss the two which are relevant to our work: **One-hot Encoding**, and; **Embeddings**.

In the **One-hot Encoding** technique, the categorical data is represented with a binary vector of size equal to the number of different values of a given attribute, where each position in the vector corresponds to a categorical value. When encoding the data, the obtained result corresponds to a vector of zeroes with a one in the position of the categorical value received as input.

However, this technique has some disadvantages. First, one-hot encoded features rely on sparse vectors, which are memory intensive. Second, there is a problem of "meaning" in one-hot encoded features, that is, we do not have a way of giving meaning to each feature value such that we can make relations between the values. For example, it is logical that the colour "red" is closer to the colour "orange" than it is to "black". However, with one-hot encoded features, we cannot observe this, since the original semantics of the categorical value were not maintained in the conversion to a numerical value.

The **Embeddings** technique addresses these problems. This technique is composed of two steps. In the first step, called tokenization, each categorical value of an attribute is seen as a token and a token map is created to obtain an index for each token. This token map maps every possible categorical value of the attribute to an integer, which is its corresponding index. In the second step, performed by an embedder, a lookup table is indexed using the value's index to obtain its final vector representation, called an embedding.

The embedder maps each feature value that was tokenized into a low-dimensional numerical vector (embedding), such that, the semantics of the original categorical values can be maintained as much as possible, i.e. (stating it in a very simplified manner) the embeddings that are closer in the multidimensional space should have a similar meaning, while embeddings that are farther in the multidimensional space should have different meanings. To put it simply, the embedder serves as a lookup table which maps indexes to the corresponding vector representations (embeddings).

There are several embedders, one for each tokenized feature. On the one hand, we may start off with some pre-trained embeddings, that are already "world knowledge", which the model will not need to learn, for example, in language models, it is very common to use pre-trained embeddings, to take advantage of semantic relationships between words already learnt by big language models. We can

then just fine-tune the embeddings to better fit the task at hand.

On the other hand, we may also start off with a completely random embedding initialization and just learn the embeddings from scratch together with the remaining parameters of the model.

In our work, the setup consists in starting with randomly (following a distribution) initialized embeddings. This was a setup that was not changed from the original model developed by OUTSYSTEMS since that is out of the scope of this thesis.

2.2.2 Merger

The role of the merger is to obtain a single representation for each one of the graph's components. As such, for each graph component, it grabs each one of the features of that component, received as input, and concatenates them to form a single embedding. Other merge operations could be performed (for instance, averaging, or even having an attention layer so that the merge is also learnt during the training of the model), however, it was previously verified that, for our use case, the concatenation operation worked the best, thus we kept this setup in our work. The resulting merged embedding is a numerical vector which corresponds to the representation of that component. Therefore, the output of the merger is an embedding (a vector in a multidimensional space) for each node and edge and another for the global component.

2.2.3 GNN model

The GNN model component of the pipeline is the core of the process since it receives a given graph with the embeddings for each graph component and performs several operations on the graph in order to calculate the final representation of the graph's components. Graphs have a very special property in that they are strictly defined in terms of their structure, however, their structure varies from graph to graph. As such, the learning model needs to be able to work with the structure of any graph, while also taking into account the relations between the various entities of the graph which are defined by its structure. In a graph, the entities are the nodes and the relations are the edges.

Therefore, the representation of a given node needs to take into account the information of the node itself, but also from its neighbours and from the relations with its neighbours. A way to perform this is by having each entity share messages with its neighbours which contain information about itself. Hence, the main idea behind GNNs is to pass messages through the entities of the graph such that they can update their representation based on their neighbourhood. This type of GNN architecture is called Message Passing.

Figure 2.2 describes how the information of each graph component is calculated based on the information of the other graph components. This figure represents one block of a GNN, the basic element of

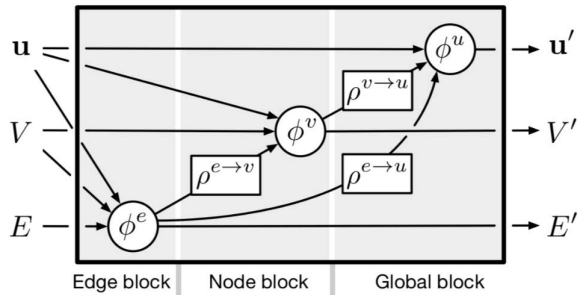


Figure 2.2: Diagram of a Full GNN block (image taken from [1])

a GNN, which updates the information for each one of the graph components based on the information of their relations. The way each component is updated can vary depending on the configuration of the GNN block, in our case, due to the configuration used, the block is named a Full GNN block.

In this type of configuration, the edges are updated using the learnable function ϕ^e first, so for each edge, the function receives as input the representations of the source node, the destination node, the edge itself and the global component and outputs the new representation of the edge. Hence, for each edge e_k we have, $e'_k = \phi^e(v_{r_k}, v_{s_k}, e_k, u)$, where e'_k is the new edge representation, v_{r_k} is the source node representation and v_{s_k} the destination node representation. After all edges are updated, each node's representation is updated using the learnable function ϕ^v, which receives as input the representations of the node, the global component and a special representation which is calculated from the aggregation of all the representations of the incoming edges of the node through the function $\rho^{e\to v}$. Thus, for each node v_k we have, $v'_k = \phi^v(v_k, \rho^{e\to v}(E'_k), u)$, where v'_k is the new node representation and E'_k are the updated representations of the incoming edges of the node. Since the representations of the incoming edges were influenced by the representation of the nodes and of the edge itself, the aggregation of the edges' representations can be seen as the passing of the messages between the nodes. Lastly, the global component representation is updated using the ϕ^u learnable function. This function receives as input the current representation of the global component and the aggregated representation of all the nodes and all the edges, calculated using the functions $\rho^{v\to u}$ and $\rho^{e\to u}$, respectively. As such, for the global component u we get, $u' = \phi^u(\rho^{v\to u}(V'), \rho^{e\to u}(E'), u)$, where u' is the new global component representation, V' are the updated representations of all the nodes and E' are the updated representations of all the edges.The aggregation functions ($\rho^{e\to v}$, $\rho^{v\to u}$ and $\rho^{e\to u}$) can be any permutation invariant function, in our case it is the elementwise *sum* function. The update functions (ϕ^e, ϕ^v and ϕ^u) are Multilayer Perceptrons (MLPs), whose parameters are learnt throughout the model training procedure.

However, it is not enough to update the information of the graph components once, that is because the information of a given entity depends on its neighbours, which in turn also depends on its neighbours' neighbours, and so on. Thus, the information needs to be propagated through the graph over several iterations. To do this we can stack several blocks to propagate the information further, for example, if

we have k blocks then each node will be able to get the information of its neighbours that are at most k-hops away. Therefore, a GNN model is simply a sequence of GNN blocks, where each layer is one block. These blocks can share the same parameters or have independent parameters.

2.2.4 Classification Head

The classification head component receives one of the graph's components as input and outputs a prediction. The classification head is a simple MLP with *softmax* as an activation function (this activation function is not required, but is used in our work), which is trained in order to improve its predictions. There are three possible classification tasks when using GNNs:

- **Node Classification** In this type of task, the objective is to classify one of the attributes of the node from its representation. One common example is classifying the category of a paper from its citations.
- **Edge Classification/Link Prediction** In this task, the objective of the model is to classify one of the attributes of an edge from its representation or to predict if two entities have a connection. One example is a recommendation system, for instance recommending new friends in a social network.
- **Graph Classification** In this task, the underlying objective is to classify the whole graph. One recurrent example is molecule classification from a molecular structure.

Depending on the desired task, a different graph component is passed to the classification head. In this work, we focus on the node classification task where the objective is to predict one of the attributes of the target node. Hence, the final representation of the target node obtained from the GNN is passed into the classification head whose output is a prediction of the desired node's attribute.

2.3 Federated Learning

Traditional Machine Learning approaches assume that the entire data set is available to the workers that build the model (in a distributed setting, and for performance reasons, different workers may access different portions of the data set but are free to access any data point). In contrast, in Federated Learning, a set of clients aim at building a common model without explicitly exchanging their data sets and, ideally, preserving the privacy of their data.

The most common approach to achieve FL consists in using a central server to orchestrate the coordination among the clients. This architecture is described by Bonawitz et al. [3]. The protocol proceeds in rounds of communication where, in each round, the server selects a set of clients to participate. When

the round starts, the server sends the parameters of the current global model to each participant. Afterwards, each participant independently trains the model received, using its own data set, obtaining a local model. The client then sends an update back to the server which reflects the changes that have been locally applied to the global model. The central server collects the updates from different clients, aggregates them in a weighted manner considering the size of each client's data set, and uses the resulting global update to derive the new global model. These rounds are repeated, possibly involving a different subset of clients in each round, until the model converges, which means the model performance stabilizes within a certain error range of the final value.

Federated Averaging (FedAvg) [4] is the underlying algorithm used in FL to train the model locally in each client and to aggregate the local updates in the central server. This algorithm relies on the Stochastic Gradient Descent (SGD) [5] algorithm to calculate the local update of the client. Therefore, in a communication round, after initializing its local model parameters with the received global model parameters, the client divides its data set, $\mathcal{D}$, into batches of smaller size, B, and performs E rounds of local computation where, in each round, it performs several updates to the local model, one update for each batch. After finishing the E local rounds, the client sends its local model parameters to the server. The server collects the parameters from the different clients and performs a weighted average of all the received model parameters considering the size of each client's data set, obtaining the new global model parameters.

FL introduces several challenges, for instance, although clients do not share their data sets explicitly, it may be possible to obtain information about the clients' private data from the client updates or even from the global model. Also, a malicious client may attempt to bias the model. Additionally, if the data each client has differs significantly from the data of the others, it may be hard to ensure that the global model outperforms the local models when trained in isolation, in such cases, we are dealing with a heterogeneous setting.

Different categories to classify FL approaches have been proposed in the literature [6–8]. One can categorize FL according to how the data is partitioned across clients, the communication architecture and the scale of the federation.

In terms of how the data is partitioned across clients, it can be categorized into horizontal, vertical or hybrid partitioning. In horizontal partitioning, data shares common features from client to client but the data subjects differ, for example, two sports teams have different supporters but collect the same type of information for each supporter. In vertical partitioning, the data features differ between clients, but the data subjects are similar, for example, a pharmacy and a bakery in a given region will probably have the same clients but will record different data about them. Finally, hybrid partitioning or transfer learning is applied when the data differs both in the subjects but also in the features, for example, a hospital from the United States and a store from Germany will have different clients and will record different information

about each client.

Regarding the communication architecture, FL can be classified as centralized or decentralized. Centralized FL corresponds to the approach described previously. In decentralized FL, no central server exists, thus the clients communicate with each other and each client can update the global model directly.

Finally, we can classify the FL approaches according to the scale of the federation as cross-silo or cross-device. In the cross-silo setting, the clients are typically organizations or data centers, hence, the number of clients is small with each one having a large amount of data and computational power. Normally, in this setting, communications and computational power are not an issue. In the cross-device setting, the number of clients is very large and clients are typically mobile devices, which have limited computational power and network connectivity, therefore, communication and computation efficiency is a key concern.

2.4 Communication Efficiency in Federated Learning

A characteristic of FL is that clients may be heterogeneous, with different computational power and available bandwidth. Kairouz et al. [6] divide the different optimizations that can be implemented to address client heterogeneity into three objectives:

- ***Gradient compression***, where the local updates from the client are compressed. Compression of updates can be performed by applying techniques of sparsification, quantization and/or subsampling. In sparsification, the parameter matrix is transformed into a sparse matrix reducing the number of parameters sent (only non-zero entries are sent). Quantization transforms the model parameters from continuous values into discrete values, reducing the number of bits for each parameter. Subsampling means sending only a part of the model. Despite being able to be combined, a few of these techniques require more computation on the client-side, and others cannot be combined with some of the privacy defence strategies that will be discussed in the next section, namely Differential Privacy, since this defence strategy adds real-noise to the updates which is incompatible with the quantization compression strategy;

- ***Model broadcast compression***, where the global model update sent to each client is compressed using the same techniques mentioned previously;

- ***Local computation reduction***, where the training algorithm is modified in order to be more computationally efficient.

2.5 Privacy of Client Data

Privacy is a major concern of FL, as one of its main objectives is to allow private client data to be used to train ML models without allowing the derivation of any data or data characteristics from the global or client updates.

However, some successful attacks showed that it is possible to leak client data, through inference attacks. Before outlining their types and possible defences, it is important to classify the different types of possible adversaries [9, 10]:

- ***Honest-but-curious server***, where the central server behaves correctly, but can try to infer information from the received client updates;
- ***Malicious server***, where the central server can deviate from the protocol, and therefore can send an arbitrary global update to the clients;
- ***Honest-but-curious client***, where a client behaves correctly, but can try to infer sensitive data from the received global updates;
- ***Malicious client***, where a client can insert arbitrary data in its data set or send an arbitrary local update to the server;
- ***Outsider***, where the attacker does not participate in the protocol but has access to the global model, either by eavesdropping communications or by using the final model.

Inference attacks try to infer sensitive information of the training data from the model updates and they can be of one of four types [10, 11]:

- ***Inference of class representatives***, where the attacker generates data samples that are not the real training data, but represent the classes of that data;
- ***Inference of membership***, where the attacker tries to infer if a given data point was used during the training process;
- ***Inference of properties***, where the attacker tries to infer some properties of the training data;
- ***Inference of training samples and labels***, or reconstruction attack, where the attacker tries to fully reconstruct the training data.

There are three commonly used approaches to protect from the above attacks, each with its trade-offs [10, 12]:

- ***Homomorphic Encryption***, where the clients leverage encryption to hide their local updates, while still allowing some operations on the encrypted data. Therefore, the local updates from the

clients are hidden and cannot be used to perform inference attacks. Different operations can be performed, however, having a larger suite of operations requires more computational power and possibly loss of utility due to some approximations. Even with a smaller set of operations, the computational overhead introduced by encryption makes it impractical in scenarios with a large number of clients;

- ***Secure Multiparty Computation (SMC)***, where the clients jointly compute a function using their own data, only having access to their private data and the function result. Furthermore, the server does not have access to client updates as these are masked in a way such that the masks from all the updates cancel out when aggregating on the server. However, it does not protect against inference attacks to the global update as SMC only protects the client updates. The main disadvantage of this approach is that it introduces both communication and computation overheads, which is detrimental in situations with a large number of participating clients;

- ***Differential Privacy (DP)***, where noise is added to the updates, either local or global, such that the sent update does not entirely match the calculated one. The noise can be inserted either by the server or the client depending on which update should be protected, global or local, respectively. However, this privacy protection comes at the cost of accuracy, since the updates do not correspond to the real computed values.

It should be noted that Homomorphic Encryption and SMC approaches mask the client updates. However, this also facilitates poisoning attacks, covered in Section 2.6, since the server cannot distinguish outlier updates from malicious updates. This is not the case for DP based approaches.

Lastly, An attacker can have access to local and global updates by eavesdropping the communication channels. Thus, secure communication channels must be used to mitigate this threat.

2.6 Security of the Model

In FL, several clients participate in each round and directly influence the global model. This opens the possibility of malicious clients performing poisoning attacks. These attacks consist of an attacker trying to modify the global model to behave in a certain way and they can be of two types [10]:

- ***Data poisoning***, where an attacker adds malicious data points that will poison the local model and consequently poison the global model;

- ***Model poisoning***, where an attacker can send arbitrary local updates that will poison the global model.

Model poisoning attacks are significantly more powerful than data poisoning attacks, since the attackers can send any desired update to the server instead of manipulating the training data. Poisoning attacks can be further divided into targeted and untargeted attacks. Targeted attacks are those where an attacker tries to make the model perform in a certain way for a certain input, without affecting the behaviour for other inputs. Untargeted attacks are those where the attacker's intent is to simply compromise the global model. In production environments, the latter have more impact since they influence the whole model utility. In terms of duration, attacks can also be determined as one-shot (only in one communication round) or continuous (during several communication rounds).

Data poisoning attacks can also be classified as clean-label or as dirty-label. In clean-label attacks, the attacker cannot modify the labels, or classifications, of its training data as desired, since the labels must be consistent with the corresponding data, whereas in dirty-label attacks the attacker can introduce new training data with any desired labels.

The defence mechanisms available to try and mitigate poisoning attacks can be of two types depending on the strategy used [13]:

- ***Robust Aggregation***, where the aggregation algorithm is improved by, for instance, selecting for aggregation only the closest update or set of updates to the median or mean of the received updates;

- ***Anomaly Detection***, where client updates considered as anomalies (or outliers) are dropped or their influence is limited since updates significantly different from the rest are more likely to be poisoned updates.

Unfortunately, there is no perfect defence strategy: some incur significant computational costs on the server-side, which can be prohibitive in some cases; some severely affect the performance of the model in heterogeneous settings since, in these settings, updates are more likely to be genuinely different, so it is more likely that correct updates are wrongfully dropped, and; most of the current defences have been shown to be vulnerable to some attacks.

It is also important to mention that Shejwalkar et al. [14] argue that, for production environments, the existent *untargeted* poisoning attacks are not effective due to unrealistic assumptions, for instance, some attacks consider a significant percentage of compromised clients (in some cases 25% to 50%), which can mean a huge number of clients needs to be compromised. Their results show that the impact of these attacks is negligible in production scenarios (less than 1% impact, with 1% of compromised clients) even with long continuous attacks. Furthermore, they also demonstrate that the least computational-intensive defence strategy is enough to protect FL in production against untargeted attacks.

2.7 Personalized Federated Learning

In an FL setting, one of the most unique characteristics is statistical heterogeneity. This particular characteristic means that each client's private data has its own specific features which can be completely different from other clients' data, and therefore the data of one client does not represent the data of the remaining clients. In this case, we say that clients have non-independent and non-identically distributed data (non-i.i.d. data). Furthermore, the number of data points in a client's data set can differ from the other clients' data set size, meaning, the data may be unbalanced in the number of data points.

Such characteristics pose a significant challenge when trying to develop global models which generalize well to all clients. In order to handle the heterogeneity of the client data, FL methods were proposed with the intent of personalizing the global model to the different clients. Tan et al. [15] propose a taxonomy where these methods can be classified at a high level as:

- ***Data-based***, which focuses on reducing the heterogeneity between different client data sets, to obtain a single global model;
- ***Model-based***, which focuses on creating models that adapt to each client.

Next, some of the sub-types of methods are explained. The first two correspond to data-based approaches and the remaining to model-based approaches.

2.7.1 *Data Augmentation*

This technique in FL consists in leveraging shared data in the training process. This data can be either directly shared by clients, or generated through a server model trained with the shared data. This strategy ensures the local updates of each client are trained with some data points that represent the overall data distribution, thus alleviating the heterogeneity of their updates. However, it leads to some performance loss as models are less personalized and also poses privacy risks as some data needs to be shared.

2.7.2 *Client Selection*

These approaches modify the server's client selection policy to favour certain clients for participation in a communication round. Therefore, clients with more heterogeneous data sets are not selected as frequently for training. However, this defeats the purpose of FL which is to learn from the different clients and introduces issues of fairness in client participation.

2.7.3 *Meta-learning*

This method is model-based and it consists in exposing the global model to several similar tasks making it learn new similar tasks quickly and efficiently.

2.7.4 *Regularization*

This method works by regularizing the client updates to limit the impact of highly heterogeneous updates, hence, creating a better generalized global model and improving convergence.

2.7.5 *Clustering*

In this technique, clients are grouped into clusters based on the similarity of their local updates, such that a model is trained for each one. However, this approach normally incurs high communication and computation costs, and it requires additional cluster management logic.

2.7.6 *Multi-task Learning*

In this method, a model is trained to learn several different tasks, trying to capture relationships between each task, in order to generalize through what it has learnt from each one. In FL this approach considers each client as a task, therefore, each one trains its own model, which is improved through the collaboration with other client models.

2.7.7 *Parameter Decoupling*

This technique decouples the global model into two parts, a representation, or body, and a classifier, or head. The body extracts the features from the data and the head uses those features to output a classification. Depending on the algorithm, one of them is global and the other is local and specialized to each client. The clients can train both sub-models, but the updates exchanged refer only to the global part, therefore, the communication efficiency is improved. This approach allows each client to have its own personalized model composed of the global shared part and the local specialized part.

2.8 Federated Learning with Graph Neural Networks

The use of FL to build GNNs is a recent research topic, in this section, we outline some of the research done. In terms of privacy preservation, Jiang et al. [16] propose an FL system for a graph classification

task that interprets video frame sequences in a semi-supervised setting, which leverages an SMC protocol, secure aggregation (covered in Section 3.2). Furthermore, Zhou et al. [17] propose an FL system for node classification tasks in vertical FL settings, using local DP.

In terms of personalization, Wang et al. [18] propose an FL system for node classification in a semi-supervised setting, leveraging meta-learning for vertical FL. Xie et al. [19] propose an FL system for graph classification in a vertical FL setting using clustering to group similar clients through their updates.

Zhang et al. [20] view each client as having a specific local subgraph and attempts to generate a global classifier for node classification in all subgraphs through FL while allowing the subgraphs to be connected between them.

All the mentioned approaches either focus on a different task other than node classification or assume a different FL setting than ours (detailed in Section 4.2).

2.9 Federated Learning Frameworks

Federated AI Technology Enabler (FATE) [21] is an FL framework for production environments. It offers privacy protection through homomorphic encryption for horizontal FL, meaning that both the server and the clients only get access to the aggregated result. Moreover, FATE allows for customization of some FL steps, such as the aggregation algorithm. However, this means that the framework does not protect against some other types of attacks, covered in Section 2.5. Also, FATE was mainly developed for cross-silo environments as the usage of encryption makes it too computationally expensive to run on client devices. Finally, the implementation of new FL algorithms requires directly changing the source code, which makes it not as flexible as desired.

Flower [2] is an FL framework that tries to provide tools for both production and research environments. It is designed for cross-device settings and allows for device heterogeneity, since it is client agnostic, meaning it does not depend on the client's hardware or software. Furthermore, it is designed in a way that introduces low overhead on the client and is scalable so as to be able to run on a significant amount of devices. Moreover, it is designed for flexibility allowing customization of several things, for instance, the averaging algorithm and training strategies. However, by default, Flower does not offer any kind of defence strategy, therefore, it is susceptible to attacks, nonetheless, mechanisms like Secure Aggregation and DP have been implemented and can be easily added.

FedGraphNN [22] is a framework specifically created for the development of FL using GNNs. Built as an extension of the *FedML* [23] framework, it allows each client to have its own graph structure since the global model is not affected by the clients' graph structure. The underlying structure used by the framework for the GNNs is the message-passing structure, covered earlier. The framework is very flexible allowing the definition of new models, datasets and FL algorithms. Also, It is mainly intended to

be used for the cross-silo setting. Moreover, it supports some privacy and security mechanisms such as DP and Secure Aggregation. However, it is a framework which is more research-oriented.

Summary

This chapter has provided the required background for our work, including ML, GNNs, and FL. We have also addressed some key challenges in federated learning, namely, communication efficiency, privacy, security and model personalization. Finally, we have covered some of the frameworks which help in the development of an FL system. In the next chapter, we describe some of the most relevant systems that, using different approaches, address the challenges above.

3

Related Work

Contents

In this chapter, we present the main works that address the previously mentioned Federated Learning challenges. Therefore, Section 3.1 explores communication efficiency systems, Section 3.2 covers privacy attacks and defenses, Section 3.3 covers poisoning attacks and defenses, and Section 3.4 addresses personalization systems.

3.1 Systems Addressing Communication Efficiency

3.1.1 *Structured and Sketched updates*

Structured and Sketched updates [24] are two optimizations proposed for client-to-server communications. *Structured update* forces the local updates to have a specific structure, that is, it forces them to be a low-rank matrix or a sparse matrix, meaning fewer parameters need to be sent in either case. *Sketched update* is an optimization where the local update is compressed after local training using subsampling or quantization techniques.

3.1.2 *Federated Dropout*

Federated Dropout [25] is a technique to reduce the costs of server-to-client communications by training updates for sub-models of the whole model. This technique can be further enhanced with the compression techniques mentioned previously. Therefore, the server first constructs a sub-model of the global model and compresses it through quantization, sending the compressed update to the client. The client decompresses the received update and trains it with its local data, calculating the local update. Then, the client compresses the obtained local update and sends it to the server, which upon receiving all the client updates, decompresses them and performs an aggregation to obtain the global update.

3.1.3 *Communication-Mitigated Federated Learning*

Communication-Mitigated Federated Learning (CMFL) [26] changes the clients' training algorithm by making each client only share local updates that represent relevant changes to the model. In the proposed algorithm, the clients determine if their updates are relevant by comparing their local update with the received global update. If the update is deemed irrelevant, then it is discarded. Through this procedure, clients with smaller contributions to the model in some communication rounds can be spared from sending their model updates.

3.2 Systems Addressing Privacy

3.2.1 Privacy Attacks

There are several examples in the literature illustrating how FL can be compromised. Inference of class representatives attacks are explored by the *multi-task GAN for Auxiliary Identification (mGAN-AI)* [27] framework which leverages Generative Adversarial Network (GAN) to break the client-level privacy by generating class representatives of the client data from the local updates.

Inference of membership attacks have been demonstrated by Nasr et al. [28], where the target model is run against a data point to individually compute the model layers so as to obtain the model features. Then, through an encoder, the probability of that data point having been used to train the model is obtained.

Inference of properties attacks are illustrated by Melis et al. [29]. The proposed attack consists in using the global updates to generate both updates based on data that contains the desired property and updates based on data without the desired property. With both these updates, a classifier is trained to indicate whether a model update was trained with the given property or not.

Inference of training samples and labels attacks have been exemplified by the *Deep Leakage from Gradients* [30] attack. This attack leverages local updates to obtain private training data by generating dummy data and dummy labels and modifying them iteratively to minimize the distance between the local update and the calculated update using the dummy data.

3.2.2 Privacy Defense Systems

3.2.2.A *Secure Multiparty Computation*

SMC is one of the privacy defence strategies of which the *Secure Aggregation* [31] protocol is an example. This protocol hides the client updates through pair-wise masks relative to each pair of participating clients. The protocol ensures both the server and clients can only see the aggregated result, hence, protecting client updates. It considers both an honest-but-curious server and a malicious server. For simplicity, we only present a simplified view of the former setting. In this setting, each client generates a secret shared seed with every other client. After calculating the local update, the clients generate a mask for every client using their shared seed. Then, for every pair of clients, one of the clients adds the mask to its update while the other subtracts it, such that when aggregating all updates on the server, the masks are cancelled out and the result obtained is equivalent to aggregating the unmasked updates.

However, global updates are still vulnerable to inference attacks. Furthermore, the communication costs increase as additional information needs to be sent to/from the server as well as the computation costs since clients need to add the masks to their local updates. Lastly, as it hides client updates, it is

not compatible with poisoning defences (Section 2.6).

3.2.2.B *Homomorphic Encryption*

Homomorphic Encryption is another possible defence strategy. Phong et al. [32] propose a system leveraging additive homomorphic encryption to protect from an honest-but-curious server. In this mechanism, the clients establish a common key pair between them that is used to encrypt and decrypt the updates exchanged with the server, which never sees updates in plaintext. Each client first receives the encrypted model parameters, decrypts them and trains the model locally, obtaining the local update. The update is then split into several parts which are encrypted individually and sent to the server. The server receives each client's parts and adds each one to the corresponding part of the encrypted model.

However, this approach assumes a cross-silo FL setting where the participants are honest, thus it does not take into account inference attacks performed by clients. Furthermore, due to encryption, the computational costs for the clients increase significantly as well as the communication costs, since more bits need to be sent for each update. This increase in computational and communication costs makes it a prohibitive approach for a cross-device FL setting.

3.2.2.C *Differential Privacy*

DP is the last defence strategy. Geyer et al. [33] propose a system to protect the global model updates by changing the server's averaging procedure. Hence, the server first clips the received updates using the Euclidean distance, such that it limits the amount of information learnt from each update. Then adds Gaussian noise to the sum of all clipped updates, further limiting the information gained from the aggregated updates. Finally, the result is normalized by the number of participating clients to obtain the noised global update.

This strategy protects from attackers performing inference attacks to the global updates, but requires the server to be trusted, since it has access to the local updates. Furthermore, since noise is added to the global update, the performance of the model decreases. Nonetheless, as demonstrated by McMahan et al. [34], the performance can be improved by increasing the number of clients. This technique is, therefore, advantageous in cross-device FL where there is a significant number of clients. Moreover, contrarily to other strategies, since client updates are not hidden from the server, poisoning defence techniques can be used.

3.3 Systems Addressing Poisoning

3.3.1 Poisoning Attacks

Several poising attacks have been identified in the literature. Data Poisoning attacks are illustrated by Tolpegin et al. [35] through a label-flipping attack which is a dirty-label targeted data poisoning attack. In this attack, a group of malicious clients purposely change the label of their training data to a given target to make the global model misclassify the input of a given source class to the target class. For example, the attacker could make the global model classify an image of a cat as a dog by changing the label of all the cats in its training data to a dog.

Model Poisoning attacks have been explored by Bagdasaryan et al. [36] through a targeted model poisoning attack, named the backdoor attack, where the global model is modified such that it performs in a certain way in an attacker task, but keeps performing well on the task it was developed for. The attack functions by sending a local update that represents the difference between the attacker's intended model and the global model multiplied by a scaling factor to make sure the backdoor survives the aggregation on the server.

3.3.2 Poisoning Defense Systems

3.3.2.A *Krum*

Krum [37] is a robust aggregation defense system that tolerates up to *f* colluding malicious clients through the definition of an aggregation rule. Therefore, when aggregating the local updates from *n* clients, the server computes the set of the $n - f - 2$ closest updates to each one of the received updates. Then, a score is calculated for each update based on its closest set. Lastly, the server selects the update with the least score which is used to obtain the global model update, as this is the closest to the majority of the other updates.

However, *Krum* has its flaws. Firstly, it limits the number of malicious clients to $n/3$, therefore, it is not flexible. Secondly, it is computationally expensive as the server needs to calculate the closest updates for each one of the received client updates, which can become costly with a large number of clients.

3.3.2.B *Foolsgold*

Foolsgold [38] is an anomaly detection defense system to mitigate targeted poisoning attacks performed by sybils, that is, several colluding clients controlled by the same attacker. It works on the underlying assumption that sybils submit similar updates since they are working towards the same goal. Therefore, it checks for similar updates and reduces the learning rate of such updates, effectively limiting the attacker's influence.

One key feature of this approach is that it does not limit the number of malicious clients since the server will increasingly degrade the learning rate the more similar updates appear. However, this approach needs to be augmented with other systems such as *Krum* to protect from other types of poisoning attacks.

3.4 Systems Addressing Personalization

In this section, some of the proposed systems to support personalization are presented. Our main focus is on systems using the parameter decoupling technique, however, some other systems are also presented. The systems presented, are some of the most recent or most cited in the literature. We characterize each system according to: the number of models produced, as single model (S) or multi-model (M), and; the local training, as joint (J) or disjoint (D), depending on if the body and head are trained jointly or not. Moreover, multi-model approaches are also classified according to the custom part of the model for each client, as full model (F), body (B) or head (H).

3.4.1 *Personalized Federated Averaging*

Personalized Federated Averaging (Per-FedAvg) [39] can be classified as a single-model joint-training (SJ) algorithm based on meta-learning, built on top of the Model Agnostic Meta Learning (MAML) framework. MAML consists in finding an initialization model (the meta-model) that performs well in a new task after an update has been performed to it, the update consists of a few steps of gradient descent. Hence, Per-FedAvg intends to find the initialization model by changing the initial goal of the FedAvg algorithm, which is to find a model that performs the best for all clients, to finding a model which can be easily adapted to perform well on each client. In order to achieve this new goal, Per-FedAvg works similarly to FedAvg, but performs more local computations to fine-tune the model to each client's data. Note that even though after fine-tuning the initialization model, we obtain multiple models, the main goal of the algorithm is to find this single initialization model, therefore, we classify it as a single model approach.

Per-FedAvg makes it easy for new clients to develop their models, only needing to fine-tune the initialization model. However, it requires more client computation for fine-tuning; has worse performance than other algorithms (from [40]), and; is more costly in terms of communication, since the full model is exchanged.

3.4.2 *FedProx*

FedProx [41] can be classified as a single-model joint-training (SJ) framework based on regularization. This framework deals with both client heterogeneity and statistical heterogeneity by changing the clients'

local objective. It works like FedAvg in the sense that, in each communication round, the server selects a set of clients to participate, and sends the global model parameters to each one. However, in FedAvg the number of local rounds are fixed and the same for each client, and if a client does not complete such local rounds within a given time frame, its updates are skipped. Therefore, to account for client heterogeneity, *FedProx* allows each client to perform a variable amount of local rounds, and submit the work done to the server as long as the work done is within a certain threshold of the final objective; this threshold is variable for each client. Secondly, to deal with statistical heterogeneity, it includes a proximal term to the local client objective, which controls how much the client can deviate from the received global update, impeding local updates that are too heterogeneous, and ensuring convergence of the global model.

FedProx is very flexible since it allows variable work for each client. However, it exchanges the full model so it is costly in communications and it restricts the client updates, meaning the global model will not be adequate leading to poor performance in very heterogeneous settings.

3.4.3 *FedU*

FedU [42] can be classified as a multi-model joint-training full-model (MJF) multi-task learning algorithm which leverages Laplacian regularization and takes into account client relationships. Multi-task learning aims at trying to find relationships from other tasks and use the knowledge from each one to improve. In the case of FL, *FedU* proposes that each client learns an individual model and uses the other related client models to improve its own model, that is, there is no global model. The server is the entity responsible for adapting each client model according to its relationships. Therefore, it follows a different approach from FedAvg and Per-FedAvg as the server maintains both the model parameters for each client as well as the relationships between the clients and their strengths. The framework works as follows, firstly the server selects a set of clients to participate in a certain communication round and sends to each participating client their current model parameters. Then, each client trains locally the model with its local data and sends the new parameters of its model to the server. The server, after receiving the client updates, applies a regularizer to each participating client's received model parameters, which takes into account the models of the client's relationships, such that it learns from other clients' models. The new client model parameters are sent to each client when they participate in a future round.

This approach allows each client to have its own local model while still benefiting from weighted collaboration, meaning a client can choose how much to learn from each other client. However, it requires the server to store a large amount of information as it needs to store both the clients' models and their relationships, so it might not be suitable for settings with a large number of clients. Furthermore, the computation on the server increases substantially, as the server needs to update each participating client's model in every communication round. Also, the communications are costly since the full model

is exchanged.

3.4.4 *Federated Learning with Personalization Layers*

Federated Learning with Personalization Layers (FedPer) [43] can be classified as a multi-model joint-training head-custom (MJH) algorithm based on parameter decoupling. Figure 3.1 summarizes all the algorithms based on the parameter decoupling technique which will be covered next, specifically, Figure 3.1(a) summarizes the FedPer algorithm, the joint chains indicate the training is joint. In this approach, the model is divided into two parts: the body, or representation, which contains the first layers of the model that are shared among all the clients; and the head, or classifier, which contains the last layers of the model that are personalized for each client. The body is trained through collaborative training using the FedAvg algorithm, whereas, the head, is trained locally with each client's data, being specialized to each one. Therefore, only the body is exchanged between the clients and the server and not the full model, thus, the server sends the current global body parameters to the clients and aggregates the received local body updates, calculating the new global body. The clients receive the global body from the server and join it with their local head to form the local client model. Then, the local model is trained normally through a few local rounds of SGD, being subsequently decoupled such that the trained body is sent to the server, while the trained head remains at the client.

This approach has better communication costs than the previous three since it only exchanges the body and it seems to converge in a few communication rounds. Nonetheless, it is not flexible since the local training is performed jointly. It is possible to improve performance by adapting the head to the received body through fine-tuning of the head before the training of the local model, however, this implies more computation.

3.4.5 *Local Global Federated Averaging*

Local Global Federated Averaging (LG-FedAvg) [44] can be classified as a multi-model joint-training body-custom (MJB) algorithm that takes the opposite approach from FedPer. Figure 3.1(b) summarizes the algorithm. It instead personalizes the body, in order to extract the high-level features of the data of each client, and shares the head, so as to develop a classifier that works for every client. By operating on local representations, the global model is significantly smaller since only the head needs to be exchanged which, typically, is smaller than the body, meaning the communication overhead is lower. In a communication round, after receiving the head from the server, the clients train their local model (obtained by combining the local body with the received global head) jointly, that is, the same SGD step updates both the body and the head. After performing all the local computation rounds, the clients send their updated heads to the server, which performs a weighted average of the received client heads,

considering each client's data set size, to obtain the global head for the next communication round.

Although LG-FedAvg achieves better communication efficiency than FedPer, it seems that its performance is lower than FedPer's, according to the experimental evaluation made by Collins et al. [40], indicating that personalizing the head may be better than personalizing the body.

3.4.6 *Federated Representation Learning*

Federated Representation Learning (FedRep) [40] can be classified as a multi-model disjoint-training head-custom (MDH) approach similar to FedPer but with a slight change. Figure 3.1(c) summarizes the algorithm, the separate chains indicate the training is disjoint and the number next to each chain indicates the order of training. The authors argue that the results from the centralized deep ML suggest that data shares a common feature representation and the heterogeneity resides in the classifications. Therefore, FedRep, similarly to FedPer, divides the model in two, where the body is shared in order to try to generate a common representation across the clients. The main difference from FedPer is in the client's local computation; while FedPer trains both head and body jointly in the same step of SGD, FedRep fully trains the head first and then the body, and each one can have its own number of training steps. This approach makes it simpler for new clients to join the system since they only need to develop a personalized head as they can use the global body already developed. Furthermore, the algorithm converges faster with more participating clients, making it suitable for a cross-device setting.

FedRep achieves better performance than both FedPer and LG-FedAvg, according to the experimental evaluation made by the authors. Also, FedRep is flexible since it allows setting a different number of local rounds for training the head and the body, however, this flexibility comes at the cost of more local computation. Furthermore, FedRep typically has more communication costs than LG-FedAvg since the body is exchanged.

3.4.7 *Federated Averaging with Body Aggregation and Body Update*

Federated Averaging with Body Aggregation and Body Update (FedBABU) [45] can be classified as a multi-model disjoint-training head-custom (MDH) algorithm based on parameter decoupling. Figure 3.1(d) summarizes the algorithm. Similarly to FedRep, FedBABU shares the body, such that, a good representation of the data is collaboratively created by the clients. The authors studied the FedAvg algorithm to understand why an increase in the performance of the global model does not necessarily mean that fine-tuning it further increases the performance. They came to the conclusion that aggregating the head introduces unnecessary noise to the global model, as the classification is a specificity of each client. Therefore, FedBABU leverages a shared fixed global head to train the body in each client, focusing on creating a good generalized global representation. Then, and only during evaluation, the

head is fine-tuned to each client. Although the training phase only generates a single model, similarly to Per-FedAvg, we consider FedBABU to be a multi-model approach since its intent is not to create a single model but to develop multiple models composed of a single global representation and custom heads.

The authors demonstrate the importance of fixing the head in FedBABU by showing it performs better than FedRep when there is only one round of head personalization in FedRep, since in such case the only difference between the two algorithms is the training of the head in FedRep. Furthermore, in the empirical studies performed, FedBABU achieves better performance than the other mentioned personalization algorithms in most of the training settings.

However, FedBABU's fine-tuning is performed during evaluation, which is not ideal since it requires a few more computation rounds. Therefore, it would be preferable to have an algorithm that produces a model fully ready for inference after the training phase. Also, since the model is fine-tuned from a fixed head for every client, all the clients begin fine-tuning the head from the same initial point. Since all the clients perform the same number of rounds of fine-tuning, it can lead to some not being able to personalize their head sufficiently, therefore, a head trained throughout the training phase would be more personalized which could maybe lead to some performance gains. Moreover, if the head was trained during the training process it would continuously be adapted to the changing body.

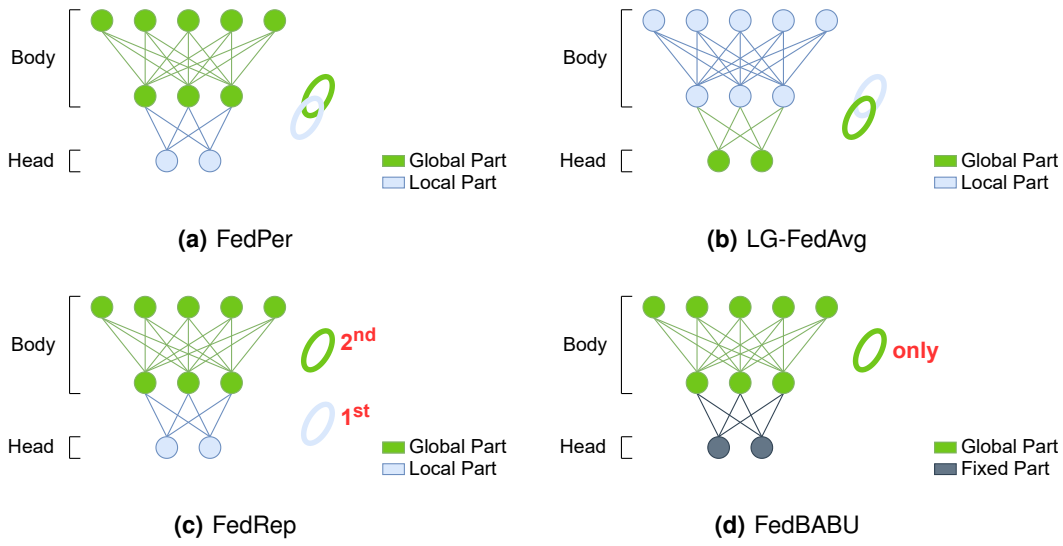


Figure 3.1: Model according to some parameter decoupling algorithms

Table 3.1 summarizes the algorithms mentioned. In terms of notation: N is the number of clients; E the number of local training rounds; E_H the number of local head training rounds and; E_B the number of local body training rounds. However, some remarks need to be done: the number of SGD steps refer to a single communication round and assume there is no batching of the data set; and the number of steps for Per-FedAvg varies depending on the approximation used.

Table 3.1: Comparison between the different FL personalization algorithms.

Algorithm	Taxonomy	Strategy Used	Number of Models	Exchanged Part	Custom Part	SGD steps per Client	Local Training Procedure	Fine-Tuning (FT Part)
Per-FedAvg	SJ	Meta-learning	1	full model	-	$2E$ or $3E$	full model	required during training (full model)
FedProx	SJ	Regularization	1	full model	-	E	full model	optional after training (full model)
FedU	MJF	Multi-task Learning	N	full model	full	E	full model	optional after training (full model)
FedPer	MJH	Parameter Decoupling	N	body	head	E	full model	optional after training (full model)
LG-FedAvg	MJB	Parameter Decoupling	N	head	body	E	full model	optional after training (full model)
FedRep	MDH	Parameter Decoupling	N	body	head	$E_H + E_B$	head first (w/ global body) body last (w/ trained head)	optional after training (full model)
FedBABU	MDH	Parameter Decoupling	N	body	head	E	body only (w/ fixed head)	required after training (head or full)

Summary

This chapter presented some of the systems proposed in the literature to solve the challenges of FL mentioned in the previous chapter. Therefore, systems to improve communication efficiency, provide privacy guarantees and mitigate poisoning attacks have been described. However, the main focus of this thesis resides in the model personalization challenge, thus some systems based on several techniques that allow model personalization have also been highlighted.

4

Federated Learning in OUTSYSTEMS

Contents

In this chapter, we discuss how Federated Learning can be used to train Machine Learning models in the context of the *Service Studio* platform. In Section 4.1, we introduce the problem and how FL can be of use and we also define our goals. We then present the OUTSYSTEMS FL setting in Section 4.2. In Sections 4.3 and 4.4, we propose two different hybrid approaches, *FedHybridAVGLG* and *FedHybridAvgLGDual* which combine different FL algorithms into a single algorithm. In section 4.5 we present the FL algorithms we selected for our study and in Section 4.6 we explore the implementation process of these algorithms and our proposed hybrid approaches. Finally, in Section 4.7 we compare the different proposed hybrid algorithms.

4.1 Motivation and Goals

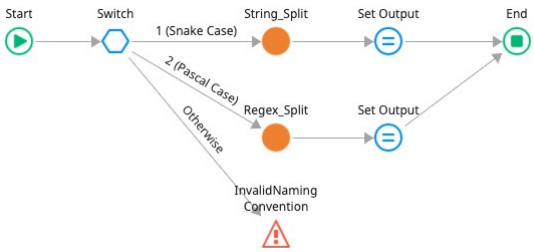


Figure 4.1: *Service Studio* Action Flow for splitting a string into multiple tokens from a given naming convention.

Service Studio is a platform developed by OUTSYSTEMS, which intends to help programmers develop their applications in a simple way, without the need to write code. As such, the programmers simply need to create a chain of actions, called an action flow, which represents the logic of the application. As an example, Figure 4.1 shows an OUTSYSTEMS Action Flow for splitting a string formatted in a given naming convention. The flow leverages a "`switch`" action to select the initial string naming convention format, either snake case (condition 1) or pascal case (condition 2), otherwise, it raises an exception. For the snake case, it first uses a "`server action`", which runs logic on the server side, to split the string by "_" and sets the output with an "`assign`" action. For the pascal case, first a "`server action`" is performed to split the string by capital letters and then the output is set. For example, for the input string "FederatedLearning" in the Pascal naming convention, this flow outputs "Federated Learning".

Action Flows can be modelled as graphs and can be classified as directed weakly connected graphs. The nodes represent the actions and are connected through edges, which represent the flow between two actions. Each edge is directed, meaning that there exists a flow relation between a source node and a destination node. Each node has its own attributes which represent characteristics or features of the action. For example, all nodes have a kind that indicates the type of the action, e.g., "`switch`", "`assign`", "`if`", and so on. Edges also have attributes that represent the characteristics of the flow, for example for a "`switch`" action one of the edge attributes indicates the condition the edge corresponds to. In an

Table 4.1: Comparison between models for different clients.

	Number of Action Flows	Accuracy (%) Local Model	Accuracy (%) Centralized Model
Client A	47,711	75.41	65.79
Client B	60	24.14	58.62

action flow, there cannot be self-loops or parallel edges, there must be exactly one "`start`" node, where the flow begins and only has outgoing edges, and the flow finishes in an "`end`", or "`raise exception`" node, that only has incoming edges.

ML is used in this platform to give recommendations to the users whenever the user tries to add a new action by suggesting some possible next actions to be added to the action flow. These suggestions are obtained using a ML model based on GNNs (this kind of model is covered in Section 2.2). In particular, the model's objective is to predict one of the nodes' attributes: the node "kind".

Currently, this model is trained in a centralized fashion, that is, the OUTSYSTEMS clients need to share their data with a centralized server, such that, a model can be trained on the data from all the clients. However, this approach has two setbacks. Firstly, it requires the clients to share their personal data which may contain sensitive information, this raises privacy concerns and may prevent the clients from sending their data to the centralized server. Since not all clients send their data, the model is trained with fewer data, which might negatively affect the performance of the obtained model. Secondly, the obtained model is shared across all the clients and, as a result, the predictions obtained by the model are the same for every client, thus, they might not be the most adequate as there is no personalization of the predictions. Another more naïve approach would be having each OUTSYSTEMS client develop their own local model, trained only with each client's data. However, this would require each client to have enough data to be able to train its own model, which is not always the case.

Table 4.1 highlights the advantages and disadvantages, in terms of the quality of the recommendations, for the usage of local models in relation to the usage of a single global model, calculated from the data of 881 clients, resorting to two distinct clients. Client A has a long usage history of the platform, therefore, it already has a large data set. As such, it has enough data points to construct a local model which offers great accuracy and is specialized to its business model. For this client, the usage of a centralized model does not amount to any benefits, on the contrary, it leads to some loss of specialization. On the other hand, client B is relatively new to the platform, thus it has a small data set. This client clearly benefits from the usage of a centralized model.

One way to allow the creation of collaborative models without the need to share private data is by using FL. Furthermore, as seen in Section 2.7, there are approaches which focus on personalizing FL models to each client. This allows the creation of models which are trained with the data of various clients without sharing client data, while also being personalized to each client's data. In this work, we

focus solely on the personalization approaches based on the parameter decoupling technique. We also propose two possible approaches, each combining two different algorithms from the ones covered in Section 3.4, we deem these algorithms hybrid algorithms (the reasoning behind the chosen algorithms for the proposed approaches will become clearer in Section 5.3).

4.1.1 Goals

Our goal is to perform an experimental study of some of the algorithms for personalizing FL in the context of *Service Studio*, while also proposing possible hybrid algorithms which combine the use of other FL algorithms and decide which algorithm to use on a per-client basis based on the amount of data of each client. The objective behind these hybrid algorithms is to be able to gather the benefits of both used algorithms with the end goal of achieving similar or better performance than the centralized models and the local models.

4.2 Federated Learning Setting

Before detailing any proposed FL hybrid algorithm, it is important to properly characterize the environment for which such an algorithm will be developed. In our setting, the clients are the servers of the organizations using the OUTSYSTEMS's *Service Studio*.

Since every organization stores all the information about each one of their action flows and the features stored for every action flow are the same for every organization, our setting, can be characterized as horizontally partitioned. Furthermore, we assume the data for each client not to be representative of the data of the other clients, since each one has its own unique action flows. This means that our setting is heterogeneous, therefore, we are dealing with non-i.i.d. data. Furthermore, the data is unbalanced in terms of the number of data points, since each client can have a significantly different number of data points in its data set.

Moreover, we assume a centralized setting where an OUTSYSTEMS server acts as the coordinator. Finally, as stated before, the clients of our FL protocol are the servers of the organizations, thus, we are working in a cross-silo setting. Having enterprise servers as end clients means that the computation requirements are more relaxed than those of traditional FL (which often relies on mobile devices). Additionally, since these machines are normally connected to enterprise networks, we can also assume that the network connection is more reliable than that of mobile devices, meaning the communication restrictions are also more relaxed in our setup.

4.2.1 Ensuring Privacy and Security

Even though we do not tackle privacy and security matters in our work we opted to characterize our environment and suggest some defences for possible future work. However, before suggesting any defence strategies to be used, it is important to first define the threat model assumed. Since the centralized server is controlled by OUTSYSTEMS, we assume it to be trusted, removing the necessity to protect the sent client updates if secure communication channels are used. The clients, on the other hand, can be both honest-but-curious, or malicious. Lastly, we also assume eavesdroppers exist and can obtain both local and global updates if the communication channels are not secure.

In terms of the security and privacy of the proposed hybrid algorithms, we argue that they do not introduce new security or privacy threats in comparison to FedAvg, since the attack surface is the same for both. The only differences between the two reside in how the local computation on the client is performed and in what parts of the models are aggregated, both of which do not increase the attacker's power.

In terms of possible defence mechanisms, and keeping in mind that these are just suggestions which are not meant to completely mitigate every attack, but rather try to difficult the attacker's attempts as much as possible, without severely affecting the utility of the model. We suggest *TLS* to be used for secure communication channels to prevent eavesdroppers; Server-side Differential Privacy in order to add noise to the global update such that clients do not have access to the fully denoised aggregated update and; an anomaly detection mechanism to try to prevent poisoning attacks, for instance, *Foolsgold* [38], which can also be enhanced by a robust aggregation algorithm.

4.3 FedHybridAvgLG

The algorithm ***Fed****erated* ***Hybrid*** *Fed****Avg LG****-FedAvg (FedHybridAvgLG)* is an hybrid algorithm which attempts to merge the FedAvg and LG-FedAvg algorithms by resorting to the FedAvg algorithm if the client has a low amount of data and resorting to LG-FedAvg if the client has a large enough data set for personalization. Smaller clients, who do not have sufficient data for personalization, leverage the FedAvg algorithm as it generates more general models since the full model is shared by all clients. Larger clients, which are those who have enough data for personalization, leverage the LG-FedAvg algorithm as it allows personalization by specializing the body of the client. The pseudocode for this approach is presented in Algorithm 1.

4.3.1 Small Clients

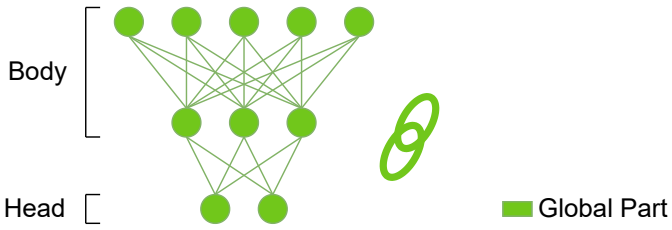


Figure 4.2: Example of a model for smaller clients in the *FedHybridAvgLG* and *FedHybridAvgLgDual* algorithms.

Figure 4.2 illustrates the scheme of a model for a small client in the *FedHybridAvgLG* algorithm. For smaller clients, which do not have enough data to personalize the model, the algorithm *FedHybridAvgLG* works exactly the same as the FedAvg algorithm. Thus, in each communication round, after receiving the model parameters, the smaller clients train the body and the head of the model jointly to obtain the trained local model. Afterwards, the client update is sent to the server, this update contains the parameters of the full trained local model.

4.3.2 Large Clients

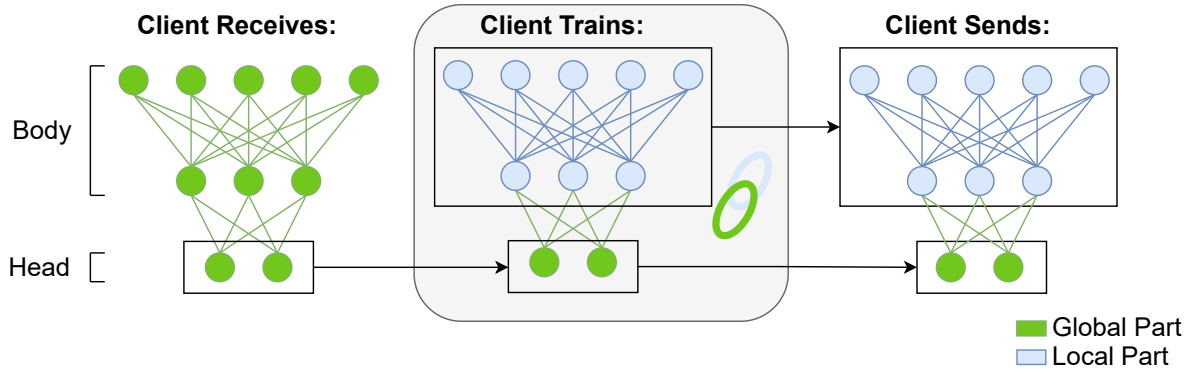


Figure 4.3: Example of a model for large clients in the *FedHybridAvgLG* algorithm.

Figure 4.3 illustrates the scheme of a model for a large client in the *FedHybridAvgLG* algorithm. For these clients, which have a large enough data set to personalize a model, *FedHybridAvgLG* leverages the LG-FedAvg algorithm. Hence, in each communication round, after receiving the model parameters, the clients only update their model head keeping their local body (instead of the whole model as in small clients) and then train both the local body and the received head jointly to obtain the trained local model. Afterwards, the client update is sent to the server. The difference between this algorithm and LG-FedAvg resides in the local update sent by these clients to the server. The local update of the clients is not solely

composed of the head of the model as in LG-FedAvg, it instead contains the parameters of the full local model, composed of the local body and the local head which was trained from the received global head.

4.4 FedHybridAvgLGDual

Fed*erated* ***Hybrid*** *Fed****Avg*** ***LG****-FedAvg* ***Dual*** *Model (FedHybridAvgLGDual)* is an approach different from the previous one, that requires the larger clients to calculate two different models (hence the dual in the name). The pseudocode for this approach is presented in Algorithm 2.

4.4.1 Small Clients

The procedure for smaller clients is exactly the same as in the *FedHybridAvgLG* algorithm. So the smaller clients simply receive the full model parameters from the server, train the model locally and send back the trained parameters. Figure 4.2 illustrates this procedure.

4.4.2 Large Clients

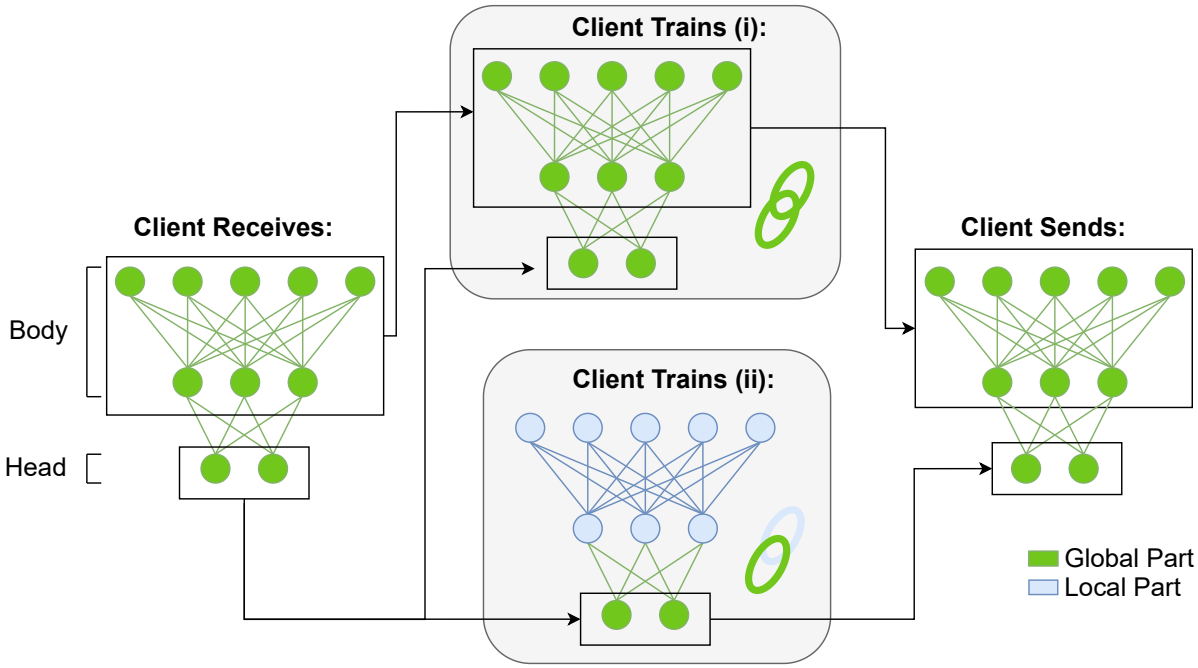


Figure 4.4: Example of a model for large clients in the *FedHybridAvgLGDual* algorithm.

When it comes to larger clients, this algorithm leverages two models, one based on the FedAvg algorithm and one based on the LG-FedAvg algorithm, Figure 4.4 illustrates the procedure for this algorithm. (i) After receiving the global model parameters, the client trains the full model received jointly to simulate the FedAvg algorithm. (ii) Then, the client keeps a local body which is trained jointly with the global head extracted from the full model parameters received from the server. After training both models, the client sends the body trained from the first model with the head trained from the second model to the server, keeping the local body from the second model.

4.5 Federated Learning Algorithms Selection

In our work, we focused on FL algorithms based on the parameter decoupling personalization technique. Hence the algorithms we selected for our study were LG-FedAvg, FedRep and FedBABU. Additionally, we also studied the FedAvg algorithm since it is the base FL algorithm.

The choice of the algorithms LG-FedAvg and FedRep allows us to study the influence of having the body local (in LG-FedAvg) or global (in FedRep) and the head local (in FedRep) or global (in LG-FedAvg). Furthermore, these two algorithms also allow us to evaluate the influence of training the head and the body jointly, that is, both are updated in the same SGD step, as in LG-FedAvg, or separately, as in FedRep, which trains the head first and only then trains the body.

Notice also, that we did not include the FedPer algorithm, although we implemented it (Section 4.6). Such a decision had to be made since we could not evaluate every algorithm as it would become too expensive, therefore, we had to opt not to study FedPer. This is due to this algorithm being fairly similar to the FedRep algorithm, once it has the body global and the head local, only differing from FedRep in terms of the client training procedure, since it trains the model jointly. However, LG-FedAvg already performs joint training and is the only algorithm to personalize the body and exchange the head, thus it would only make sense to exclude FedPer and keep both FedRep and LG-FedAvg.

Finally, FedBABU has a different approach from all the previous algorithms, allowing us to study the influence of training the body with a fixed head as well as the influence of fine-tuning the model before evaluation since it is the only algorithm that requires it.

4.6 Implementation

4.6.1 Selecting the Framework

In order to implement the various FL personalization algorithms, we opted to use the *Flower* framework, covered in Section 2.9. We decided to use this framework, since it is a very flexible framework that is easily adaptable to several models and so is not only restricted to GNNs, also it is relatively simple to modify both the server behaviour on aggregation and the local training procedure of the clients. Furthermore, it has good documentation as well as various examples of different implementations of some FL algorithms. Moreover, it is a framework which is also oriented for production environments, being very scalable, which facilitates the transition to production if OUTSYSTEMS ever desires to implement FL in production. Lastly, the one downside of *Flower* is it does not have any security and privacy defence mechanisms by default, however, there are some mechanisms which are already implemented and can easily be added.

4.6.2 *Flower* Framework

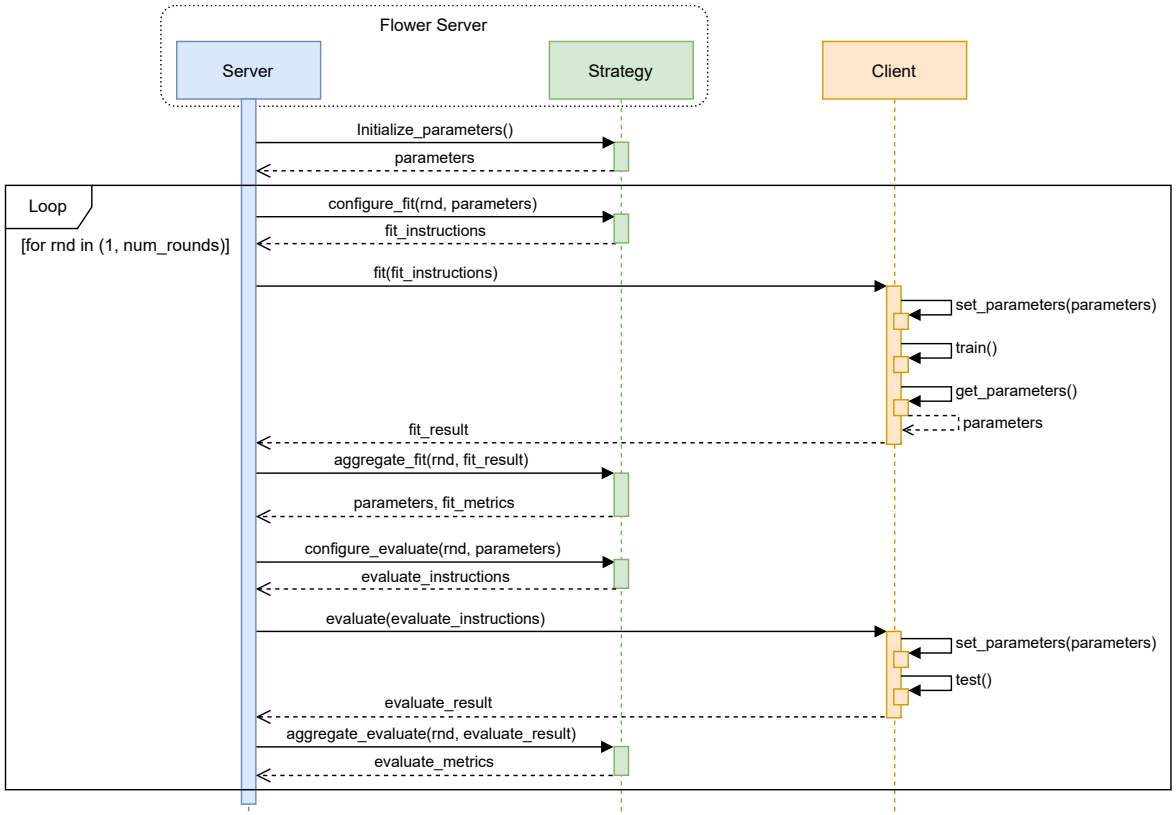


Figure 4.5: Sequence diagram of the *Flower* framework for a single client.

In terms of the implementation, as mentioned in the Section 4.6.1, we opted for the *Flower* framework in order to facilitate the implementation and evaluation of the FL algorithms. Figure 4.5 demonstrates the timeline of events of the framework during the FL training using a single client (it can be easily extended to multiple clients). In this framework, the *Flower* server is divided into two components, the "Server" and the "Strategy", the former is responsible for the coordination of the training/evaluation process and dealing with the communications with the clients, while the latter is responsible for the implementation of the logic of the various server duties, such as parameter initialization, client selection, training/evaluation instructions for the clients and result aggregation.

The execution first begins with a *initialize_parameters* call from the "Server" component to the "Strategy" component which returns the initial model parameters to be used in the first FL communication round. Afterwards, the "Server" executes a number of communication rounds defined by *num_rounds*, which is a configuration parameter. In each one of these rounds, the "Server" first obtains the training instructions to be sent to the clients by calling *configure_fit* of the "Strategy". This call selects the clients to be trained in that communication round and returns the training instructions (*fit_instructions*) for these clients. The training instructions contain the current global model parameters and some training configuration parameters to be used by the clients. Then, the "Server" sends the training instructions to each "Client" through the *fit* call. After the training is complete the results of the training containing the trained model parameters and training metrics are returned to the "Server" (*fit_results*), which aggregates all the received results by calling the *aggregate_fit* routine of the "Strategy" component, obtaining the new global model parameters and the training aggregated metrics (*fit_metrics*).

After performing the training, the new global model parameters are evaluated. As such, the "Server" obtains the evaluation instructions from the "Strategy" component by calling *configure_evaluate*. Similarly to *configure_fit* this call selects the clients for evaluation and only returns the instructions for these clients. The evaluation instructions contain the model parameters and possibly some test configuration parameters. Then, the "Server" sends the evaluation instructions to each selected "Client" through the *evaluate* call. After performing the model evaluation, the evaluation results are sent back to the "Server", which aggregates the received results by calling *aggregate_evaluate* to the "Strategy" component. This procedure is then repeated for *num_rounds* rounds.

In terms of the "Client" component, if a *fit* call is received from the "Server", the "Client" sets its model parameters to the ones received in the *fit_instructions* using the *set_parameters* call and then trains the model by calling *train*. After training, it gets the model parameters by calling *get_parameters* and returns the *fit_results* to the server. If an *evaluate* call is received, the model parameters are set to the ones received in *evaluate_instructions* and then the test is performed by calling *test*. Afterwards, the results of the test are sent back to the server (*evaluate_results*).

When implementing a new FL algorithm, both the "Strategy" and the "Client" components need to be

defined. We will now cover the implementation for each of the federated algorithms, which is available in the following GitHub repository: github.com/OS-danielfranciscolopes/FL-Personalization.

4.6.3 Strategy

In terms of the "Strategy" component, we used the pre-implemented *Flower* strategy for the FedAvg algorithm and only redefined some of the calls for each algorithm, as the logic for most calls is the same. We firstly defined a strategy which redefines the *initialize_parameters* method so as to initialize the model parameters on the server since, by default, the parameters are initialized from one of the client's parameters. Also, for the FedBABU fixed head, we used the same initialization used by Oh et al. [45] in their experiments, which is the He (uniform) initialization, the default initialization of *PyTorch* [46] (the ML framework used). We then defined a strategy which redefines the *configure_fit* and *configure_evaluate* methods to store the IPs of the selected clients for each round, such that we can keep records of which clients participated in the training/evaluation of each communication round. Lastly, we defined three different strategies based on the part of the model that is aggregated by the server.

Full Aggregation This strategy is equivalent to the FedAvg strategy, however, we had to redefine the *aggregate_fit* method, since *Flower* does not store the global model of each communication round. Thus, we redefined this method to store the global model after aggregating the received client model parameters. The algorithms FedAvg, *FedHybridAvgLG* and *FedHybridAvgLGDual* use this strategy since these aggregate the full model (notice that for *FedHybridAvgLGDual* both the smaller and larger clients send a full model even though for the larger clients the body and the head are derived from separate models).

Body Aggregation This strategy is used in the FedRep, FedBABU and FedPer algorithms. Although these algorithms only aggregate the body in the server, they require the head of the model to be initialized equivalently for each client, as such, the server needs to not only send the global body parameters, but also the initial head parameters in both the training and evaluation instructions. Therefore, we had to redefine the *configure_fit* and *configure_evaluate* methods to add the head parameters to the aggregated body parameters, which are received as an argument. Similarly to the previous aggregation strategy, the *aggregate_fit* method was also redefined in order to save each round's model.

Head Aggregation This strategy is used in the LG-FedAvg algorithm. Similarly to the previous strategy, this algorithm requires the body of the model to be initialized equivalently for each client, as such, we need the server to not only send the global head parameters but also the initial body parameters

in both the training and evaluation instructions. Hence, we had to redefine the *configure_fit* and *configure_evaluate* methods to add the body parameters to the aggregated head parameters, which are received as an argument to the call. Similarly to the other aggregation strategies, the *aggregate_fit* method was also redefined in order to save each round's model.

4.6.4 Client

When it comes to the "Client" component, when implementing a new client *Flower* needs the definition of the *set_parameters*, *get_parameters*, *fit* and *evaluate* methods. The *train* and *test* methods are not required by the framework, these are the methods used for training and testing the model which were already developed by OUTSYSTEMS for the centralized setting and were reused as the logic to train and test the model remains the same. Therefore, we first defined a base client which is the implementation of FedAvg and for each other algorithm, we simply redefined some methods. We will now cover the "Client" implementation for each algorithm.

FedAvg Since this algorithm shares the whole model, the *set_parameters* method simply sets the whole model to the received parameters and *get_parameters* returns the parameters of the full model. The *fit* method works as described in Figure 4.5, so it first calls *set_parameters*, then the *train* method and lastly *get_parameters*. For the *evaluate* method it first calls *set_parameters*, then it might call the *train* method, if fine-tuning is required, and, lastly, a call to the *test* method is then performed

LG-FedAvg In order to implement the LG-FedAvg algorithm small changes had to be made. The *get_parameters* and *set_parameters* methods had to be redefined to only return and set the head parameters, respectively.

FedPer Contrarily to LG-FedAvg, where *get_parameters* and *set_parameters* were modified to only consider the head, in the case of FedPer, these methods were redefined to only consider the body, in accordance with this algorithm's local training procedure.

FedRep Similarly to FedPer, *get_parameters* and *set_parameters* were redefined to only consider the body. Also, the *fit* method was redefined to call the *train* method twice, the first time it only trains the head and the second time it only trains the body.

FedBABU Similarly to LG-FedAvg, the *get_parameters* method was redefined to only return the body parameters. Also, the *set_parameters* call was redefined to set the body and the fixed head appropriately. Furthermore, the *fit* method was modified to use the fixed head and only train the body when calling

the *train* method. Lastly, the *evaluate* call was redefined to perform fine-tuning before calling the *test* method.

Hybrid "Client" This implementation serves as the base implementation for the hybrid algorithms. It simply sets a client as being small or large based on the total data points in the client's data set and in a threshold (defined in the next section). Thus, it does not implement any algorithm.

FedHybridAvgLG This hybrid algorithm is a combination of both FedAvg and LG-FedAvg, as such, only the *set_parameters* method needed to be redefined to set the full model parameters for smaller clients or only the head parameters for the larger clients.

FedHybridAvgLGDual In this case, *set_parameters* had to be modified, such that, for smaller clients it performs like FedAvg and for larger clients it maintains a temporary body (used in the FedAvg training), and sets the model head as in LG-FedAvg. *get_parameters* also had to be redefined, for smaller clients it is equal to FedAvg, but for larger clients it needs to return the trained temporary body (from FedAvg) and the head of the local model (from LG-FedAvg). The *fit* method for the smaller clients is the same as for FedAvg, however, for larger clients, two calls need to be made to the *train* method: one where the temporary body (containing the global body) is trained with the received global head (as in FedAvg) and; one where the local body is trained with the same received global head (as in LG-FedAvg). Since the *evaluate* method is the same for both FedAvg and LG-FedAvg, no redefinition is needed.

4.6.5 Client-Size Categorization

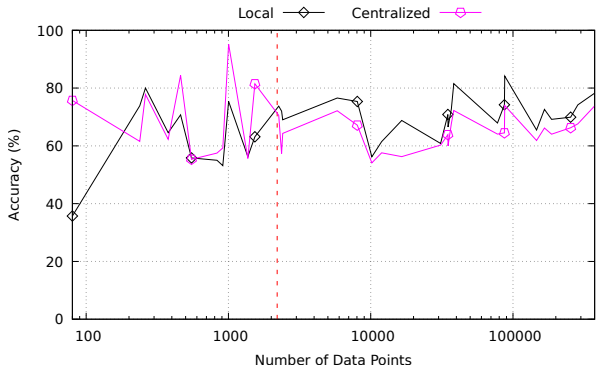


Figure 4.6: Accuracy of local and centralized models by the number of total client data points.

Hybrid algorithms combine two different FL algorithms, having each client perform one or the other depending on a given threshold. In order to define the threshold for considering a client small, we used the data from Figure 4.6. This Figure illustrates the average validation accuracy from the last 5 training

rounds of a total of 30 rounds for the local models and the centralized model, for each one of 33 selected clients. We considered the average validation accuracy over the last five rounds of training because, by then, the model should be more stable which means we should be able to gather a more representative picture of the average accuracy of the model than if we were to use another measure, for instance, the maximum accuracy of the 30 rounds, where we could get an outlier result which would not represent the model accurately. The procedure for choosing these clients is described in Section 5.2. The horizontal axis is in logarithmic scale. From this graph, we can identify a point where the clients start to have enough data to personalize a model to their use case, thus starting to prefer using local models instead of the centralized model, which is more general.

In this graph, we can see a point at approximately 2200 data points (marked in the figure with a red dashed line) where to the left of this point the centralized model generally achieves better performance (except for a couple of points) and to the right, the local models' performance surpasses the centralized model performance. Hence, to the left of this point, the clients do not have a sufficient amount of data to personalize a model to their use case and, to the right, the clients start to have a large enough data set which allows them to create one specialized model to their data, such that, a more general model starts to have worse performance.

This threshold of 2200 data points draws the line between clients preferring a more general model, such as the centralized model, and a more personalized model, like the local model. Therefore, we defined this value as the threshold for considering a client as a small client and choosing which algorithm to be used in the hybrid approaches.

4.7 Discussion

The two previously proposed hybrid algorithms present significantly different approaches. *FedHybridAvgLG* does not require the larger clients to train two different models, therefore, it is less costly computationally. Also, since this algorithm maintains a local body in the larger clients which is further specialized every round and is sent to the server, it is expected that the aggregated body in the server will be more specialized than when using the FedAvg algorithm, where the clients train a global body every round. The same happens with the global head, which will also be more specialized due to the larger clients training it with a local body.

FedHybridAvgLGDual uses a more specialized aggregated head than FedAvg since the head sent by the larger clients to the server was trained with a local specialized body. Also, contrarily to *FedHybridAvgLG*, the body sent by the larger clients is not the local specialized body and is instead a body calculated from the received global body, thus the global body is more general than in *FedHybridAvgLG*. Furthermore, since the aggregation of the global head is performed with the heads from the smaller

clients, which are not as specialized as they were trained with a more general global body, it is to be expected that the aggregated heads of *FedHybridAvgLG* and *FedHybridAvgLGDual* are less specialized than LG-FedAvg.

Also, since *FedHybridAvgLGDual* requires more computation from the larger clients as they need to compute two models, the use of this algorithm only becomes viable for the larger clients if they can obtain a certain advantage from the obtained models that is not possible when using other models, either in terms of model performance or any other advantage, for instance, the suggestion of novel actions.

Lastly, we attempted to develop an algorithm which could merge FedBABU, used by the smaller clients, and LG-FedAvg, used by the larger clients. However, since FedBABU requires the body to be trained with a fixed head, while LG-FedAvg requires the head to be shared, we were not able to design an approach to combine these two algorithms without it being the same as executing them separately.

Summary

In this chapter we have introduced the use case that is studied in this thesis, the *Service Studio* platform developed by OUTSYSTEMS, as well as the environment considered for FL. Moreover, two new algorithms have been proposed. These algorithms attempt to merge some of the FL personalization algorithms mentioned in the literature, such that, the algorithm to be used by a client is chosen based on the amount of data of each client, we name these hybrid algorithms. *FedHybridAvgLG* merges FedAvg and LG-FedAvg. *FedHybridAvgLGDual* also merges the previous two algorithms but requires the computation of two models on the clients with more data.

5

Experimental Study

Contents

Table 5.1: Statistics of the number of data points of the 33 selected clients

min	max	mean	median	var	std-dev	Percentile				
						25	75	90	95	99
80	374,860	57,855	10,175	9,006,809,135	94,904	1,003	77,790	182,641	266,686	346,235

This chapter presents the results of an experimental study performed to evaluate the performance of some of the personalized Federated Learning algorithms proposed in the literature, as well as the hybrid algorithms we proposed. As such, in Section 5.1 we present the goals of the performed experiments. Section 5.2 addresses the experimental setup used during the experiments. In Section 5.3 the results of the experiment for the node kind prediction task are presented. Section 5.4 presents the results for another prediction task with a larger number of output classes. Section 5.5 contains the results of the experiment of the recommendation of new actions to larger clients. Finally, in Sections 5.6 and 5.7 we experiment with different training parameterizations.

5.1 Goals

We performed an experimental study of the several personalization FL algorithms in the context of OUTSYSTEMS so as to answer the following questions:

- What is the best-performing algorithm for the node kind prediction task?
- Does the behaviour of the algorithms change for a more complex prediction task?
- Are the FL algorithms capable of suggesting novel actions to clients with a more extensive usage of the *Service Studio* platform?
- How does varying the number of local computation epochs influence the quality of the model of the FL algorithms?
- How do different learning rates influence the model quality for each algorithm?

5.2 Experimental Setup

In order to evaluate each one of the federated algorithms, we leveraged an OUTSYSTEMS data set composed of the code developed by 881 clients. From this data set, we selected 33 clients for evaluation. Each client maintains the data relative to one organization which uses the *Service Studio* platform, that is, it keeps all the action flows of that organization.

To extract the 33 clients, the clients were partitioned according to their number of action flows. The first partition includes all the clients with less than 64 flows and all the following partitions increase

exponentially in size by a factor of 2, creating 11 partitions in total. Afterwards, 3 clients were randomly selected from each partition. Table 5.1 contains some statistics about the number of data points of the selected clients.

The evaluation was performed in the AWS cloud where each client was run on a separate instance. For all the experiments, for the federated algorithms, 30 communication rounds were performed and for each one all the 33 clients participated, that is, there was no client selection, since in the case of OUTSYSTEMS there are no communication or hardware restrictions. The local models were obtained using the data of each one of the 33 clients and the centralized model using the data of all the 33 clients in a single instance.

5.2.1 Model Performance

Since the clients' data set is balanced (Appendix B.1 contains the class distribution for the 33 selected clients), the performance of the obtained models was evaluated using the accuracy of the models in each client's test data set, that is, the percentage of correct predictions over the total predictions. In the analysis of the results we split the clients into three groups (large clients are split into intermediate and big clients):

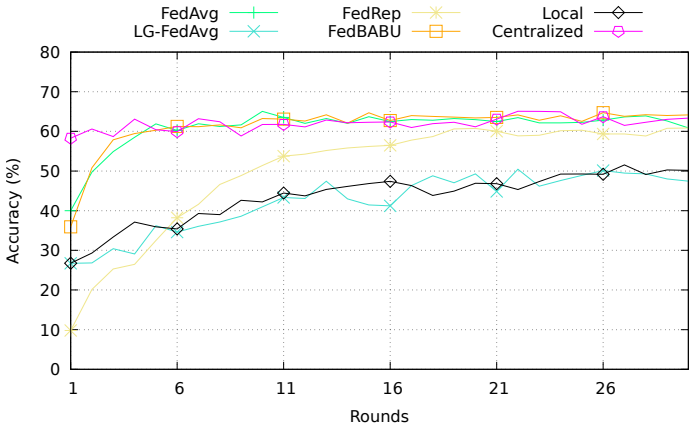
- Clients with a small number of data points (until 2200 data points, as explained in Section 4.6.5);
- Clients with an intermediate number of data points (between 2200 and 31700 data points). The value 31700 was obtained from the percentiles of the total number of data points for the 881 clients and it corresponds to the 75% percentile, and;
- Clients with a big number of data points (above 75% percentile, that is above 31700 data points).

5.3 Node Kind Prediction Task

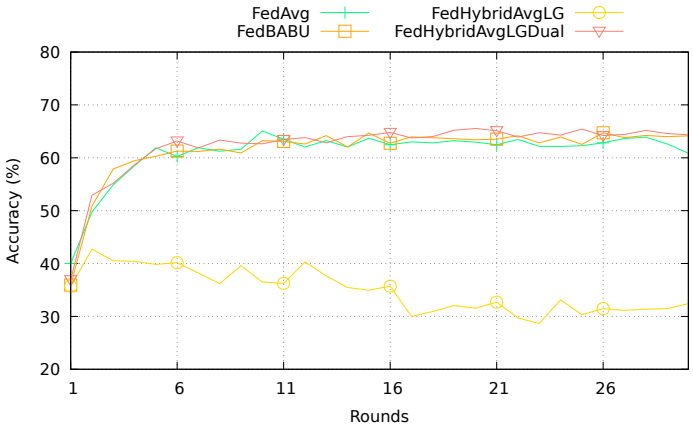
In this section, we analyse the experimental results for the node kind prediction task. In Section 5.3.1, we analyse the results for the algorithms proposed in the literature (FedAvg, LG-FedAvg, FedRep and FedBABU) and the centralized and local models. Then, we analyse the performance of our hybrid proposals in Section 5.3.2. In order to facilitate the interpretation of the results, we created two different graphs for each type of client, one which contains the literature approaches (includes the federated algorithms from the literature and the local and centralized algorithms) and another which contains the hybrid algorithms (includes the hybrid algorithms and, for comparison purposes, the literature algorithms which they intend to replicate as well as the best-performing algorithms from the literature graph for that group of clients). Finally, since we wanted to provide the same test environment for every algorithm, these results were obtained without fine-tuning, not even for the FedBABU algorithm, meaning that, for

this specific experience there is no personalization mechanism in the case of FedBABU, as such a fixed classifier was used. Hence, we performed a separate experience where we fine-tune the models for a single round before evaluation. The results of this experience are presented in Section 5.3.3. The parameters of the model used are described in Table C.1 and the experiment hyperparameters are described in Table C.3, and for the fine-tuning experiment in Table C.4.

5.3.1 Literature Algorithms



(a) Literature Algorithms



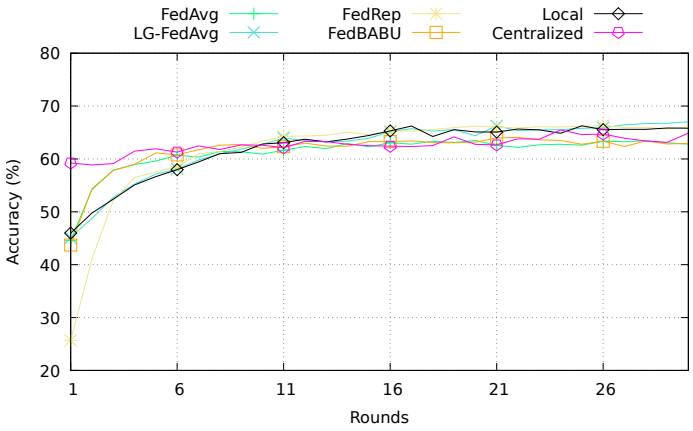
(b) Hybrid Algorithms

Figure 5.1: Accuracy of the various models for small clients for the node kind prediction task.

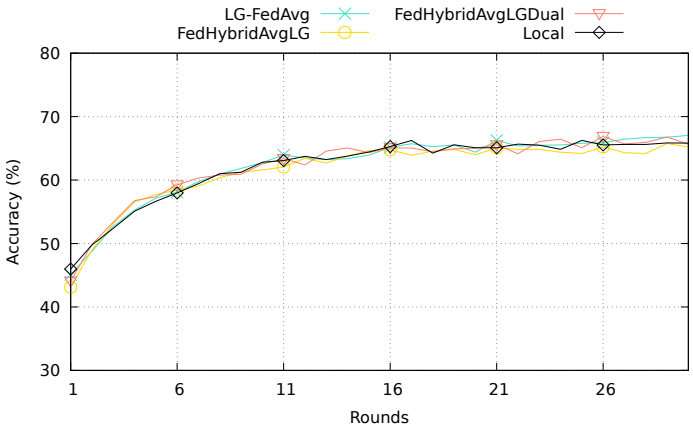
Performance for Small Clients Figure 5.1(a) shows the evolution of the average accuracy for the small clients throughout the training/communication rounds for each one of the models and algorithms. In terms of the federated models from the literature, we can see that the FedAvg and FedBABU algo-

rithms are the ones which obtain the best accuracy (with a slight advantage from FedBABU), followed by the FedRep algorithm and lastly by the LG-FedAvg, meaning that personalizing the head is preferable to personalizing the body. The LG-FedAvg algorithm achieves the worst performance, a fact that can be justified by the few data points of the client which do not allow for proper personalization of the body.

We can also check that the centralized model achieves worse accuracy than both FedAvg and FedBABU in a considerable amount of rounds. Lastly, we see that the local models are far inferior to both the centralized model and the FedAvg, FedBABU and FedRep algorithms, which shows the importance of client collaboration when clients have a small amount of data.



(a) Literature Algorithms

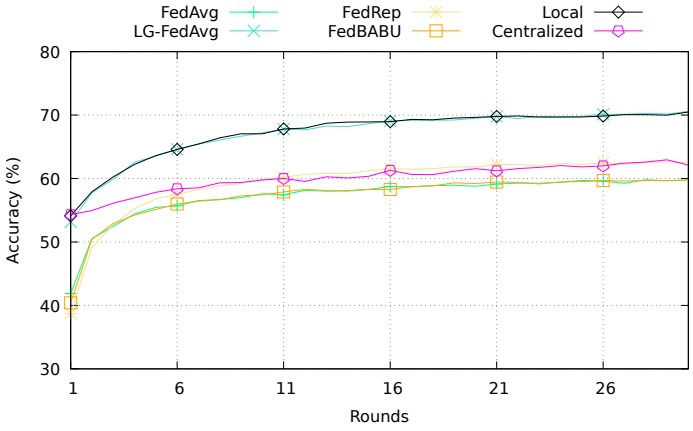


(b) Hybrid Algorithms

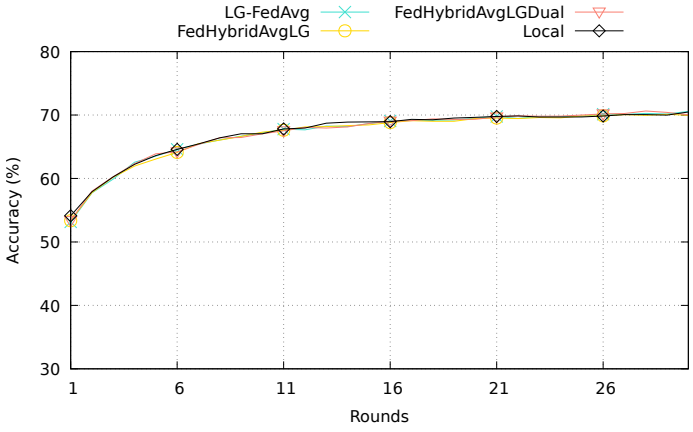
Figure 5.2: Accuracy of the various models for intermediate clients for the node kind prediction task.

Performance for Intermediate Clients Figure 5.2(a) illustrates the evolution of the average test accuracy for the clients with an intermediate number of data points. We can see greater proximity between

the accuracies of the four federated algorithms. Furthermore, the personalization algorithms are superior to FedAvg and FedBABU, in particular, the LG-FedAvg algorithm has the best accuracy, meaning that for clients with more data, personalizing the body of the model is best since the greater amount of data allows for better personalization. Also, FedBABU achieves slightly better performance than FedAvg in most of the rounds. Finally, note that towards the end of the training, LG-FedAvg and FedRep end up achieving better accuracy than both the local models (about 0.3% to 1% superior) and the centralized model (about 2% to 3% superior), with LG-FedAvg being able to surpass the performance of FedRep, meaning its body personalization is becoming more effective.



(a) Literature Algorithms



(b) Hybrid Algorithms

Figure 5.3: Accuracy of the various models for big clients for the node kind prediction task.

Performance for Big Clients Figure 5.3(a) illustrates the evolution of the average accuracy for big clients. In this case, we can see a tendency similar to the one of the intermediate clients, where the per-

sonalization algorithms are superior to FedAvg and FedBABU, which have similar performance. However, the LG-FedAvg algorithm is far superior to the FedRep algorithm, meaning that personalizing the body is the best option for clients with a lot of data. Also, towards the end of the training where the body of LG-FedAvg starts to be more specialized, the accuracy becomes slightly superior to the one of the local model (difference of 0.1% to 0.2%) and also superior to the centralized model (difference of 7% to 8%).

Discussion of Results From the obtained results we can conclude that there is no strategy that is better than the others for all types of clients. For clients with few data points, the personalization of the head turns out to be easier than the body, since the body typically has a greater number of parameters, therefore, it is more difficult to personalize. However, for these clients, either the collaboration on the full model or the collaboration on the body but using a fixed head is preferable, since the low amount of data makes personalization ineffective. The FedAvg algorithm, which trains the whole model collaboratively, obtains results very close to those of the centralized model, being superior in a considerable amount of rounds. Also, FedBABU manages to achieve slightly better results than FedAvg, which means that collaboratively training the head might introduce some noise into the model and so it is preferable to train the model with a fixed head. Hence, FedAvg and FedBABU can be an alternative to the centralized model because they allow collaboration without sharing the clients' data (contrarily to the centralized model) while managing to achieve very similar results.

As the number of data points grows (intermediate and big clients) the data becomes specific and in sufficient quantity to train, individual client models. Therefore, the centralized model becomes inferior to local models and personalization models are superior alternatives to the FedAvg and FedBABU algorithms. Also, the personalization of the body offers greater results than that of the head and actually, slightly superior to local models resulting in a difference of up to 1% in accuracy, which indicates that the collaboration on the head might help these larger clients classify some data points which are more general and less specific to the client that the local model fails to classify correctly. As such, we can conclude that for these clients, personalizing the representation is preferable to personalizing the classifier, which is somewhat surprising since the literature mentions that it is expected for the heterogeneity to reside in the classifier and not in the representation. Thus, from the results, it is possible to find a shared classifier trained collaboratively by the clients which in conjunction with a personalized representation obtains better results than those of the local model.

In environments where data privacy is a requirement, the development of a hybrid approach between the FedAvg or FedBABU algorithms (for smaller clients) and the LG-FedAvg algorithm (for bigger clients) would allow bigger clients to collaborate in the construction of a federated model which would benefit the smaller clients without sharing their data, while also receiving a small boost in model performance when

compared to local models. This reasoning is what motivated the development of the hybrid algorithms *FedHybridAvgLG* and *FedHybriAvgLGDual*, whose results will be covered next.

5.3.2 Proposed Hybrid Algorithms

Performance for Small Clients In Figure 5.1(b) we can verify the evolution of the performance for small clients for the hybrid algorithms. We can observe that *FedHybridAvgLG* is the algorithm which has the worse performance, and in fact, it gets worse over every communication round. On the other hand, *FedHybridAvgLGDual* manages to achieve the intended performance and obtain results similar to FedAvg.

Performance for Intermediate Clients From Figure 5.2(b) we can observe that *FedHybridAvgLG* does not manage to match the performance of the LG-FedAvg algorithm, which was its intended goal. Also, *FedHybridAvgLGDual* although not overperforming LG-FedAvg in every round, it manages to surpass the performance of this algorithm in some rounds and it also achieves close results in the other rounds, as intended.

Performance for Big Clients In Figure 5.3(b) the evolution of the accuracy for the big clients for each one of the hybrid algorithms can be observed. The *FedHybridAvgLG* algorithm manages to achieve similar results to the LG-FedAvg algorithm, nonetheless, it achieves inferior accuracy (difference of about 0.1% to 0.6% in accuracy). Lastly, *FedHybridAvgLGDual* also achieves similar results to the LG-FedAvg algorithm but it overperforms this algorithm in some rounds (difference of about 0.1% to 0.4% in accuracy), therefore, achieving its intended goal.

Discussion of Results The algorithm *FedHybridAvgLG* underperformed in comparison to LG-FedAvg for the intermediate and big clients, and most importantly, to FedAvg for small clients where the difference between the two is considerable and kept getting worse after each communication round. We believe this result is explained by the fact that each larger client sends its local body for aggregation. Since every round, each larger client's body keeps getting more and more specialized in its own unique way, the resulting aggregation is not of use because each body "pulls" in its own direction. This explains why the performance for smaller clients decreases as the local bodies of the larger clients get more specialized.

The *FedHybridAvgLGDual* algorithm achieved the intended results since it achieved better performance than the FedAvg algorithm for the smaller clients and similar or better performance than the LG-FedAvg model (and consequently the local model, since they have identical performance) for the larger clients. For the smaller clients, the fact that the global head is trained with the local bodies of the

larger clients means that the head becomes more specialized while also managing to remain general enough not to affect the classifications of the more general client data, leading to an improvement in performance. For the big clients, we have the opposite, as the small improvement in performance comes from the fact that the aggregated head contains the heads of the smaller clients which were trained with a more general body (remember that for smaller clients there is no local body, the full model is shared), this means that the global head is more general than the one obtained from LG-FedAvg (which trains the head with a local body in every client). This results in an improvement in the classifications of the few data points that are more general and less specific to each client's data set.

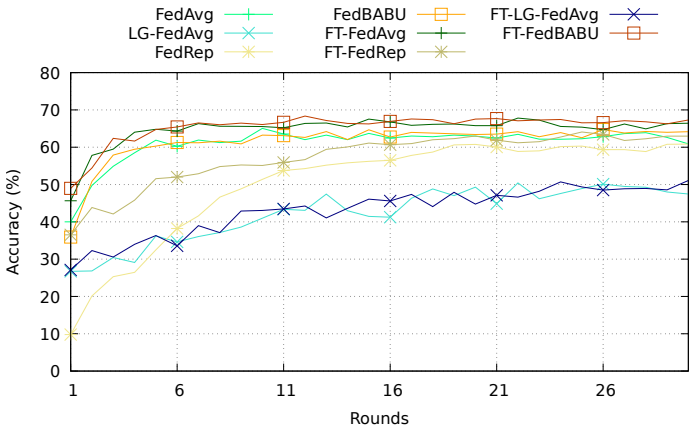
Since the *FedHybridAvgLG* algorithm did not manage to achieve the intended results, in order to save test budget, we opted not to perform any further experiments with this algorithm. As such, the remaining experiments only test the *FedHybridAvgLGDual* algorithm.

5.3.3 Fine-Tuning

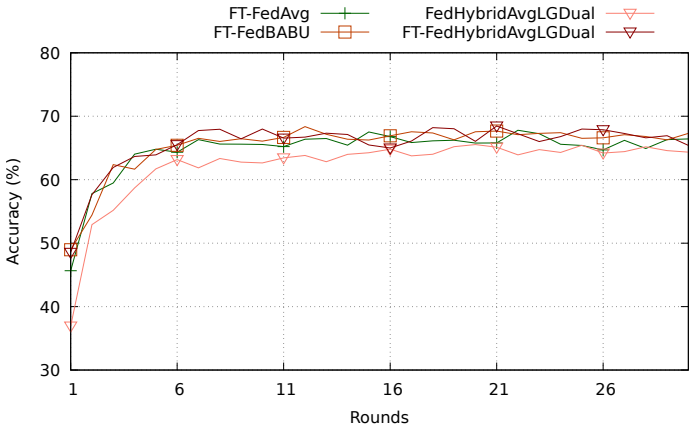
In order to test the influence of fine-tuning, we performed an experiment where we fine-tuned the models of the previous experiment for one round before evaluation. Fine-tuning was performed on the whole model. Table C.1 contains the model parameters and Table C.4 contains the experimental hyperparameters used for this experiment.

Performance for Small Clients Figure 5.4 presents the results of the accuracy for small clients when fine-tuning the models for each algorithm. Similarly to the previous results, for ease of interpretation, we split the results into two graphs, thus, Figure 5.4(a) contains the results for the literature algorithms (includes the federated algorithms from the literature and the local and centralized approaches) and Figure 5.4(b) contains the results for the hybrid algorithms (includes the hybrid algorithms and, for comparison purposes, the literature algorithm they intend to replicate as well as the best-performing algorithms from the literature graph for that group of clients). From the results we can verify that almost all algorithms benefit from fine-tuning before evaluation, meaning that a small personalization of the whole model can make the model adapt to the client's data. The only model with no improvement under fine-tuning was the LG-FedAvg model, which might indicate that personalizing the head might not be useful for these clients.

Performance for Intermediate Clients Figure 5.5 illustrates the performance for the intermediate clients with and without fine-tuning, Figure 5.5(a) contains the results for the literature algorithms and Figure 5.5(b) for the hybrid algorithms. It is possible to observe that both LG-FedAvg and our approach do not benefit from fine-tuning, which might imply that the personalization was already adequate before fine-tuning and there is no new gain in further personalization. However, the remaining algorithms



(a) Literature Algorithms

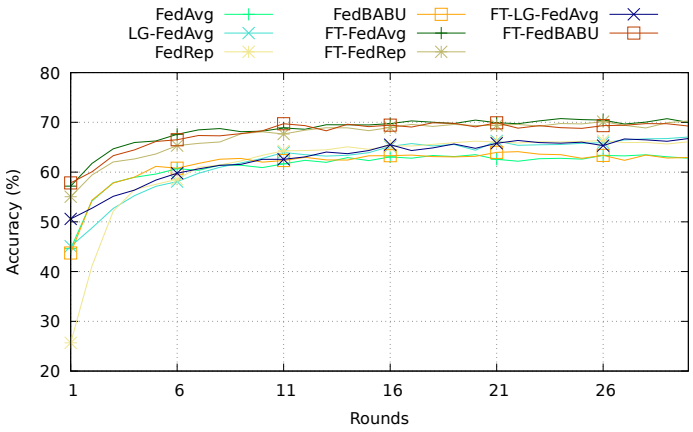


(b) Hybrid Algorithms

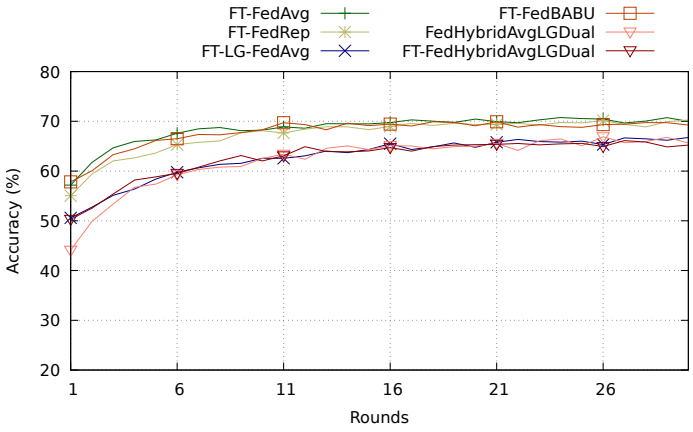
Figure 5.4: Accuracy of the various models for small clients for the node kind prediction task after fine-tuning.

manage to outperform both our approach and LG-FedAvg by quite a margin (3% to 6% in accuracy), something that indicates that intermediate clients still benefit from a more general approach which only needs an adaptation to the client data.

Performance for Big Clients Figure 5.6 contains the results with and without fine-tuning for the big clients, Figure 5.6(a) contains the results for the literature algorithms and Figure 5.6(b) for the hybrid algorithms. There are some similarities and some differences from the previous clients. As for the similarities, once again fine-tuning our hybrid approach and LG-FedAvg does not produce any gains and in some rounds, it is prejudicial for the model performance. As for the differences, the remaining algorithms do not manage to overperform the former algorithms as they did for the intermediate clients, meaning the personalization of the body manages to capture the specificities of the client data more



(a) Literature Algorithms

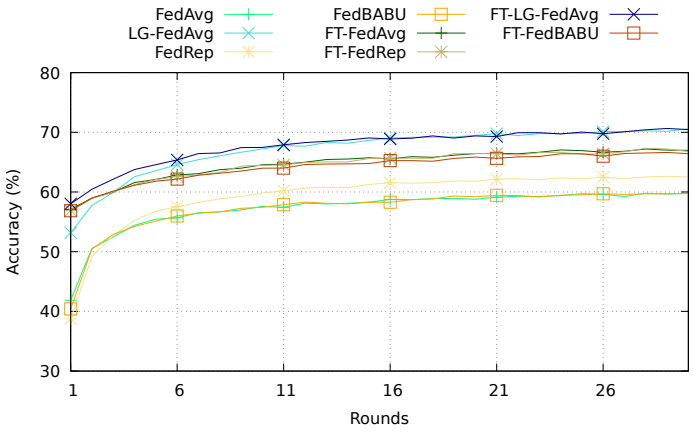


(b) Hybrid Algorithms

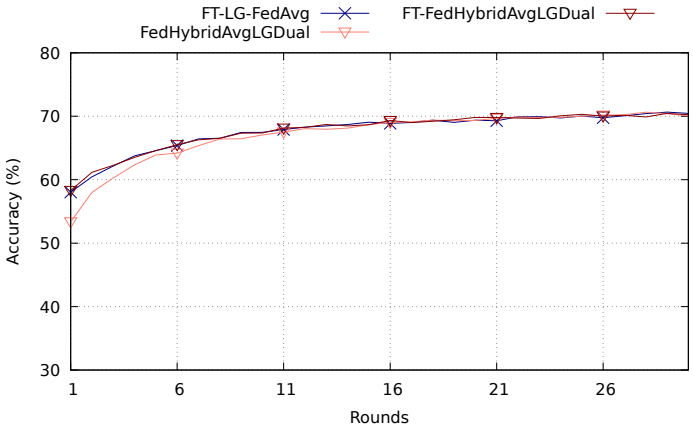
Figure 5.5: Accuracy of the various models for intermediate clients for the node kind prediction task after fine-tuning.

accurately. Also interestingly, the fine-tuning of the FedAvg algorithm achieves better performance than the fine-tuning of the FedBABU algorithm and similar performance to the fine-tuning of the FedRep algorithm, which indicates that having a personalized head might not amount to many advantages.

Discussion of Results This experiment produced some interesting results, which will now be discussed. Firstly, it is interesting to notice that fine-tuned FedBABU achieves better results than fine-tuned FedAvg for smaller clients, but worse results for intermediate and big clients. The authors of FedBABU [45] argued that training the head introduced noise to the global model, hence training the body with a fixed head. We can derive that the introduced noise in FedAvg is prejudicial for the smaller clients since it affects the generality of the model when these clients need a more general model. However, the noise results from the specific data of the larger clients, so when it is removed (as in FedBABU)



(a) Literature Algorithms



(b) Hybrid Algorithms

Figure 5.6: Accuracy of the various models for big clients for the node kind prediction task after fine-tuning.

it affects the quality of the predictions as the model becomes more general when larger clients need a more specific model.

Secondly, it is also interesting to note that the body might be the model part which best captures the specificity of the client data. By looking at the performance of the LG-FedAvg for all the clients and our hybrid approach for the intermediate and big clients (where it replicates the behaviour of LG-FedAvg), we can see that fine-tuning results in no improvement in performance, in fact, in some rounds it worsens the performance. This happens because the body has reached a point of personalization where more personalization simply has no effect (in the early rounds fine-tuning improved the accuracy, which does not happen in the later rounds) and the head by being more personalized stops classifying more general and out of the distribution client data as effectively, which can explain the drop in performance in some of the rounds after fine-tuning. Furthermore, if we look at the results of FedRep after fine-tuning and

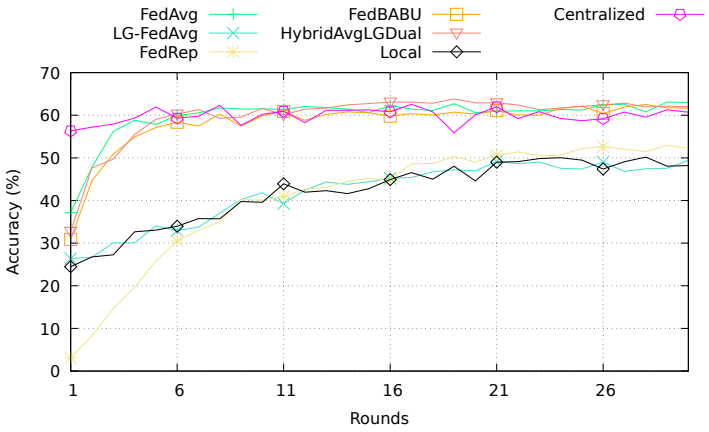
compare them to the ones of FedAvg after fine-tuning for the intermediate and big clients, we can see that fine-tuned FedAvg outperforms or matches fine-tuned FedRep which has a personalized head, hence the personalization of the body seems to be the key performance factor. Thus, it is preferable to have a more general head and a body which is personalized to the client data.

Lastly, fine-tuning FedAvg, FedRep and FedBABU in the intermediate clients achieved better performance than our approach and LG-FedAvg. This fact further proves the previous point, since the personalization of the body offered a great boost in performance in FedRep, which already had its head personalized. Moreover, it might mean that these clients, which start to have a more specific data set, still do not have enough specific data to where it is preferable to personalize the body fully instead of having a more general model which is adapted to their data when fine-tuning. Thus, if fine-tuning is desired, it may be worth developing an approach similar to *FedHybridAvgLGDual* where the intermediate clients behave like the smaller clients by performing the FedAvg algorithm and then fine-tuning it, instead of the current approach where they perform both the FedAvg and LG-FedAvg algorithm. However, the impact of the intermediate clients not participating in the LG-FedAvg part of the algorithm would have to be studied, since the global head would become more general than the one of *FedHybridAvgLGDual* which could impact the performance of the model for both the big and small clients.

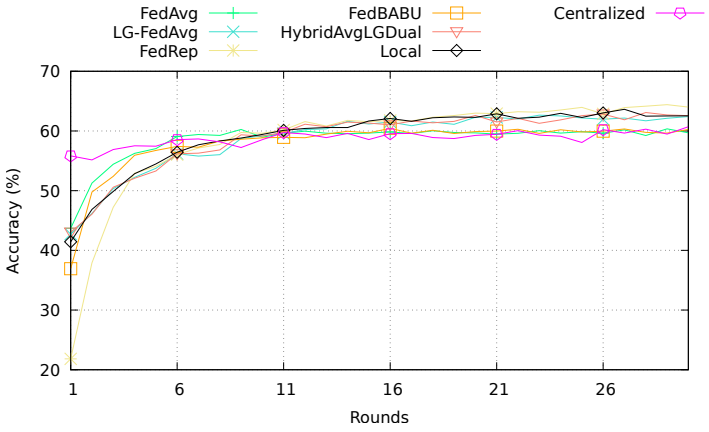
5.4 Node Subkind Prediction Task

In order to analyse the federated approaches with a more complex task, we performed an experiment where the task of the model is no longer to predict the kind of the node but, instead, the subkind of the node. The subkind is an attribute which specializes two particular node kinds, “ExecuteClientAction” and “ExecuteAction”. As such, the number of output classes required for the model is significantly bigger (from 27 to 196 output classes), meaning the heterogeneity between the clients also increases and so does the specificity of each client. Since the distribution of the classes per client is balanced, we used accuracy as a metric for this experiment (Appendix B.2 contains the class distribution for the 33 selected clients). Figure 5.7 presents the results for this experiment for each one of the types of clients. It is also worth noting that for comparison purposes the FedBABU algorithm had no fine-tuning, hence there was no personalization for this algorithm. The parameters of the model used are described in Table C.2 and the experiment hyperparameters are described in Table C.5.

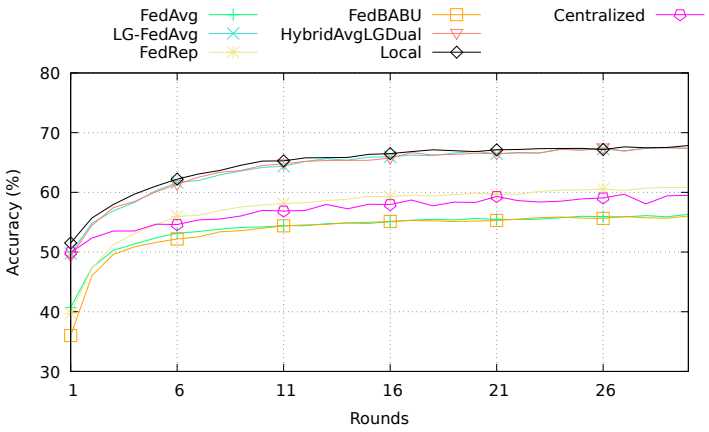
Performance for Small Clients Figure 5.7(a) illustrates the evolution of the accuracy for the small clients for the node subkind prediction task. It is possible to observe that our hybrid proposal achieves better results than the other algorithms in most of the rounds, including the centralized algorithm (difference of about 1% to 3%) being slightly outperformed by the FedAvg algorithm in the last rounds.



(a) Small Clients



(b) Intermediate Clients



(c) Big Clients

Figure 5.7: Accuracy of the various models for clients of different sizes for the node subkind prediction task.

Nonetheless, the FedAvg and FedBABU algorithms also achieve similar accuracy to the centralized model, even surpassing its performance at the later rounds. Contrarily to these algorithms, the personalization algorithms LG-FedAvg and FedRep, as well as, the local models achieve considerably lower accuracy than the previously mentioned models (difference of about 8% to 15%).

Performance for Intermediate Clients From Figure 5.7(b), which shows the evolution of the accuracy for the intermediate clients for the node subkind prediction task, we can verify that the FedRep algorithm achieves the best accuracies, followed by the local model. This indicates that the personalization of the head is preferable for these clients. However, the *FedHybridAvgLGDual* and the LG-FedAvg algorithms still manage to achieve close results to those of FedRep (1% to 2.5% difference). For these clients, personalization is a must since the algorithms FedAvg, FedBABU and the centralized model achieve the worst results.

Performance for Big Clients Figure 5.7(c) illustrates the evolution of the accuracy for the big clients for the node subkind prediction task. The results show that the local model is the best-performing model, yet our hybrid algorithm and the LG-FedAvg algorithm achieve close performance (difference of about 0.1% to 0.7%). Contrarily to the intermediate clients, the FedRep algorithm has a considerable gap in performance to the other personalized algorithms. Finally, once again for the clients with more data the centralized, FedAvg and FedBABU models achieve the worse performance, demonstrating the importance of personalization for these clients.

Discussion of Results In this experiment, since the number of output classes increased, the heterogeneity between the clients also increases, which lead to some interesting conclusions. When it comes to the smaller clients, similarly to the experiment of Section 5.3, it is preferable to share the full model, or at least not have personalization as in FedBABU (notice that no fine-tuning was performed, so there was no personalization). Furthermore, our hybrid approach seems to fit the clients better than FedAvg, which shows that having a head slightly more personalized (since the global head was trained with the local body for the larger clients) allows the model to capture more specificities of the data without affecting the performance. The personalization of the head continues to be preferable to that of the body, however, there is a considerable drop in performance for FedRep when comparing with the node kind prediction task, which is due to the high increase of output classes making the head harder to personalize.

The intermediate clients had interesting results. Contrarily to the previous experiment, FedRep achieved the best performance, followed by the local model, this indicates that, for these clients, the personalization of the head manages to capture more of the specific properties of each client's data than the personalization of the body. Furthermore, since the heterogeneity between the clients increases, a bigger gap appears between the personalized models and the centralized model. This gap was not

as big in the node kind experiment, meaning that as the heterogeneity increases so does the need for personalization.

Lastly, for the larger clients as the complexity of the problem increased, the local model achieved better results, contrarily to the previous experiment. This means that the collaborative models are not able to capture as effectively the particularities of the client data and the classifications of more general and out-of-the-distribution client data are not enough in order to compensate for the gap in specialization between local models and body-personalized models. Moreover, as the number of data points increases, the body seems to be able to capture more specificities of the data than the head (LG-FedAvg has better performance than FedRep), in contrast with the intermediate clients. Lastly, contrarily to the node kind prediction, the more general head of our hybrid approach does not produce better results than the more specialized head of LG-FedAvg, however, at the same time, it does not lose specificity since the results of both algorithms are almost equivalent.

Once again, our hybrid approach revealed to perform as intended, being able to outperform the FedAvg algorithm for the clients with smaller data sets, showing the positive influence of having a slightly more specialized head, and performing equivalently to LG-FedAvg for the clients with more data.

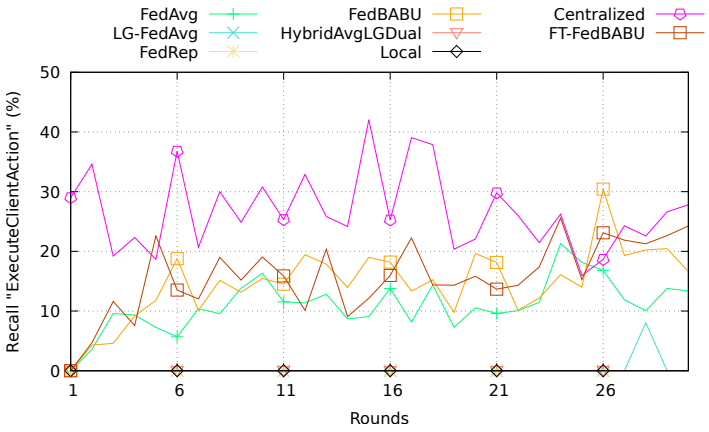
5.5 Recommendation of New Actions

As seen in the previous experiment, the local model outperformed both the LG-FedAvg and the *FedHybridAvgLGDual* for the big clients. One possible advantage of collaborative models over local models could be the recommendation of new actions because although some clients might not start using novel actions straight away, others will and since the models are created with the data of every client, the novel actions should, in theory, eventually be recommended. In order to test the recommendation of new actions to larger clients we performed an experiment where we picked two clients (one intermediate and one big) and for each client we removed one class of actions from both its training and validation data sets and kept the test data set as is.

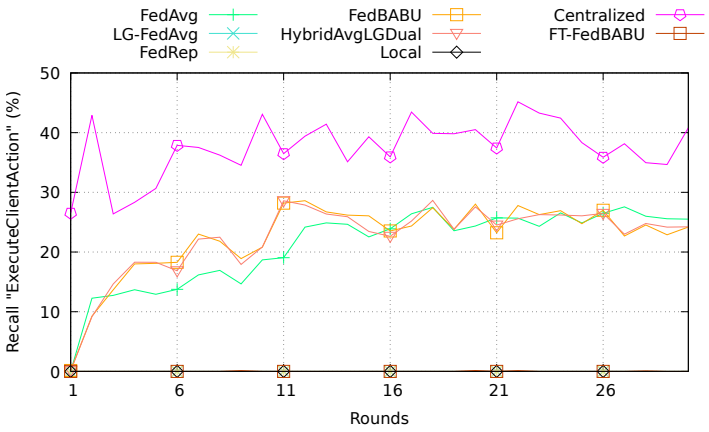
We picked the *"NRNodes.ExecuteClientAction"* class to be excluded since this was actually one of the most recent actions to be added to *Service Studio*. The two clients were selected based on the relative frequency of the *"NRNodes.ExecuteClientAction"* class on their data set since we did not want the class to be too representative as that would remove a high number of data points from the data set, but we also did not want a class with too little representation as that would mean we would not have enough data points for testing. Therefore, we opted for clients with 16515 (intermediate client) and 254206 (big client) total data points, both of which had a relative frequency of about 5% for the excluded class.

In order to evaluate the results a new metric had to be considered, as accuracy takes into account all

of the classes of the data set and we only want the accuracy relative to the excluded class. As such, the metric recall for the *"NRNodes.ExecuteClientAction"* class was used, this metric gives the percentage of elements correctly predicted from all the elements of a given class. In this experiment, we presented the results for FedBABU with and without fine-tuning, since we wanted to evaluate if fine-tuning this algorithm influenced the prediction of the new actions. Figure 5.8 contains the results for the two clients. Table C.1 contains the model parameters and Table C.6 contains the experimental hyperparameters used for this experiment.



(a) Intermediate Client



(b) Big Client

Figure 5.8: Recall of the class "ExecuteClientAction" for the various models for two clients with the class excluded from the training data.

Performance for the Intermediate Client Figure 5.8(a) presents the results for the recall of the excluded class for the intermediate client. From these results, we can clearly see that the personalization algorithms cannot capture the new class from the global part, since all of them (except FedBABU) have

a recall of 0%, with the exception of a round of LG-FedAvg where it reaches 8%. Interestingly, FedBABU achieves results which are better than FedAvg in most of the rounds, even when fine-tuning is performed. The centralized model seems to be the best model to capture and recommend new actions.

Performance for the Big Client Figure 5.8(a) shows the results for the recall of the excluded class for the big client. Once again, similarly to the intermediate client, the personalization algorithms achieve recall 0% in every round. Moreover, FedAvg and FedBABU achieve similar results and personalization of FedBABU (after fine-tuning) does not seem to affect the performance of the model. Lastly, once again, the centralized model seems to be the best model for suggesting new actions.

Discussion of Results This experiment lead to some rather interesting results. Firstly, it is clear that the aggregation performed in the federated models influences negatively the suggestion of new actions, as the centralized model achieves better recall than the FedAvg algorithm. This is most likely due to the weighted aggregation, since in the centralized model all the data points have the same weight, while in FedAvg the larger clients' data points have more weight.

Secondly, the personalization algorithms do not seem to be able to capture the new class. We believe this is due to the fact that the local part has not seen data from the excluded class. On the one hand, if the body is local then it does not know how to represent the class properly so that it can be correctly classified by the head (which is global and so as seen the class). On the other hand, if the head is local, then it does not know what to do with the features extracted by the body (which has seen the class and can extract the features appropriately), so it does not know how to classify the class. This shows a clear disadvantage of using these algorithms.

Lastly, FedBABU produced intriguing results, since both before and after personalization, this algorithm managed to correctly classify some elements of the excluded class, achieving similar results to FedAvg. This was not expected, since either the head was fixed or was personalized with data that did not contain the class. This could mean that a shared body could be enough to suggest a new class and fine-tuning was not sufficient to "erase" the knowledge of the body, however, that would mean FedRep would have to have achieved results different from 0% since it also shares the body, which was not the case. Unfortunately, due to budget constraints, we were not able to perform more experiments and try to analyse the reasoning behind these results. Nonetheless, we believe that the fact that in the other personalization approaches the local part was never in contact with the excluded class, whereas in FedBABU it either uses a fixed classifier (which has not seen any data since it was not trained), or a fine-tuned model trained from a body which had already seen the excluded class (in FedRep the head is initially trained from a body that has not seen the class) and thus adapts the fixed head to the global body, might play a role in the explanation of the results.

5.6 Varying the Number of Local Training Epochs

We also experimented varying the number of local training epochs, since we wanted to understand how this hyperparameter impacts the performance of our proposed hybrid algorithm (*FedHybridAvgLGDual*). We also evaluated the performance of FedAvg and LG-FedAvg to understand if our approach is similar to these algorithms. Table C.1 contains the model parameters and Table C.7 contains the experimental hyperparameters used for this experiment.

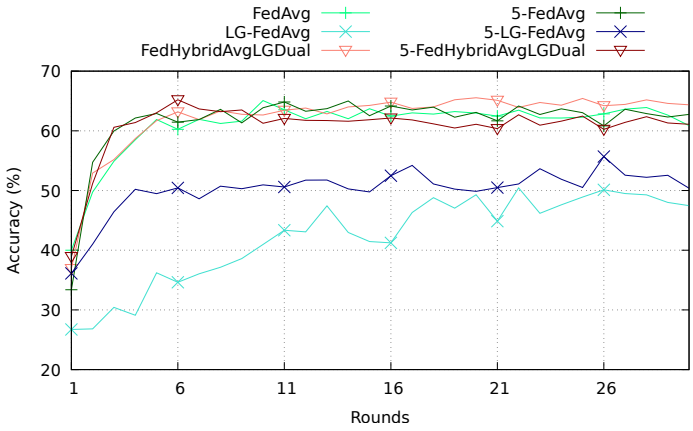


Figure 5.9: Accuracy of the various models for small clients when varying the number of local training epochs.

Performance for the Small Clients Figure 5.9 demonstrates the evolution of the average accuracy for the small clients for each algorithm and number of training rounds. From the figure, we can verify that LG-FedAvg benefits from performing more local training rounds per communication round, which indicates that specializing the body further can be beneficial. As for FedAvg, we can observe that the accuracy remains similar so more training rounds do not influence this algorithm. On the other hand, increasing the number of local rounds for *FedHybridAvgLGDual* leads to a loss of performance.

Performance for the Intermediate Clients Figure 5.10 contains the results of the experiment for the intermediate clients. For these clients, there starts to appear a performance gap in the FedAvg algorithm where performing more local rounds seems to not be beneficial, which might be due to the higher heterogeneity of the updates. On the other hand, both LG-FedAvg and *FedHybridAvgLG* seem to close the gap in performance throughout each round, eventually achieving similar accuracy. Hence, it is noticeable that more computation leads to the algorithms converging faster.

Performance for the Big Clients From Figure 5.11, we can observe the same pattern for FedAvg, where a performance gap appear after a certain point, however, the gap appears sooner than for the

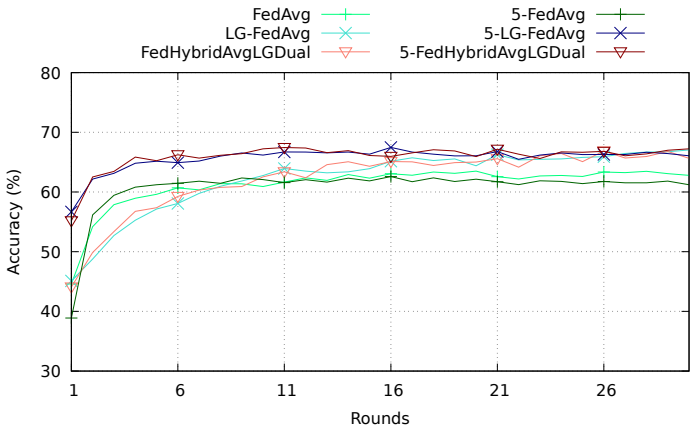


Figure 5.10: Accuracy of the various models for intermediate clients when varying the number of local training epochs.

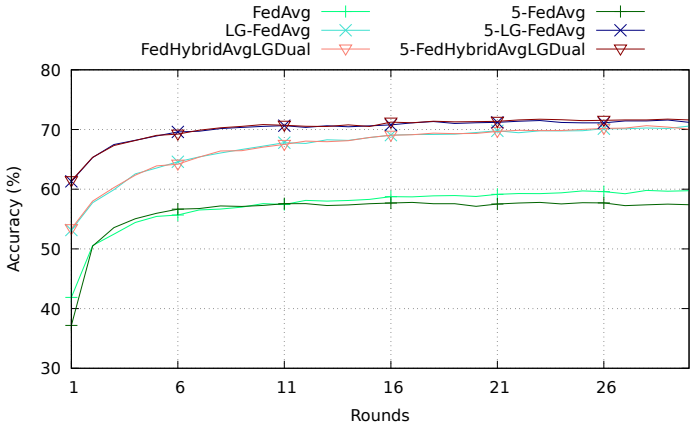


Figure 5.11: Accuracy of the various models for big clients when varying the number of local training epochs.

intermediate clients. Both LG-FedAvg and *FedHybridAvgLGDual* benefit from the more local computation, meaning that clients with more data prefer to specialize the body more, although the performance gap keeps decreasing after every round, so if more rounds were performed the performance may end up being identical, similar to the intermediate clients.

Discussion of Results From the obtained results for the FedAvg algorithm, we can clearly see that the more data each client has, the worse the gap in performance between the accuracy with one and five local rounds. This can be observed by noticing that the accuracy achieves similar results for smaller clients, but a gap of 1% to 2% appears for intermediate clients and an even bigger gap appears for the big clients, from 1.5% to 2.5%. This is due to the increase in the heterogeneity of the client updates since by performing more rounds, the local updates will be more specialized to each client resulting in a worse aggregated model. The aggregated model seems to be able to still perform similarly for less specific data (thus more general), as in the case of smaller clients, but as the data becomes more specific, the

performance worsens. However, if the heterogeneity in the updates is further increased, for example, when using our hybrid approach (remember the global head is calculated with more specialized heads from the larger clients) then the performance gap also appears in the smaller clients.

Furthermore, LG-FedAvg seems to benefit from personalizing the body more for small and big clients. In the small clients, this is to be expected since the data set is small so fewer steps are performed each round, as such, more steps allow the model to train the body more. As for the big clients, clearly, the more rounds performed, the more the model specializes in the client data. Nonetheless, it is interesting to notice that the performance gap keeps decreasing after each round, therefore it is possible that by performing more communication rounds the performance gap disappears, similarly to the results of intermediate clients where the performance eventually becomes identical, meaning the body can capture the specificities of the client data set (which is less specific than for the big clients) and more specialization results in no further improvement.

When it comes to our proposed approach, there is a clear trade-off between the increase in computation needed to perform more rounds, the loss of performance for smaller clients (3% to 4% performance decrease) and the increase in performance for the big clients (1% to 2% performance increase). This increase in performance might be a further incentive for big clients to perform federation if they are willing to accept the extra computation costs. However, it should be noted that it is possible that with more communication rounds, the performance gap for the big clients might end up vanishing, since that fact can be observed in the intermediate clients, similarly to the LG-FedAvg algorithm.

One final note about this experiment, we only experimented with 5 local training rounds as it would be too expensive to test more values, nevertheless, future work might address this and experiment with more rounds to find an ideal value.

5.7 Varying the Learning Rate

In order to understand how the performance of our proposed hybrid approach (*FedHybridAvgLGDual*) is influenced by the learning rate, we performed an experiment where we tested different learning rates: 0.1, 0.01, 0.001 and 0.0001. We also evaluated the performance of FedAvg and LG-FedAvg to understand if our approach is similar to these algorithms. Figure 5.12 illustrates the average accuracy per round for each one of the types of clients for each learning rate for *FedHybridAvgLGDual*. The results for FedAvg and LG-FedAvg are present in Figure 5.13 and Figure 5.14, respectively. Table C.1 contains the model parameters and Table C.8 contains the experimental hyperparameters used for this experiment.

Performance for the Small Clients Figure 5.12(a) contains the results for the *FedHybridAvgLGDual* algorithm for different values of learning rate. From these results, we can observe that smaller learning

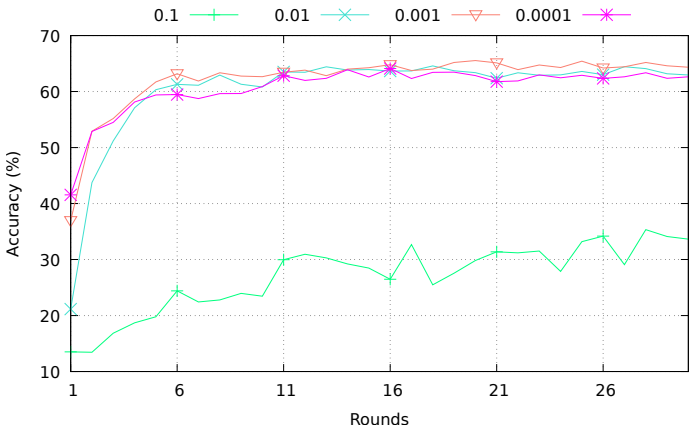
rates achieve better performances since there is a drastic performance gap from 0.1 to the other learning rates. The remaining values achieve similar results, however, 0.001 achieves slightly better accuracy. For the FedAvg and LG-FedAvg algorithms (Figures 5.13(a) and 5.14(a), respectively) all the learning rates achieve very similar performance, except for the value 0.1, where once again there is a gap to the other values, although smaller when comparing with *FedHybridAvgLGDual*.

Performance for the Intermediate Clients Figure 5.12(b) presents the performance for the *FedHybridAvgLGDual* algorithm for the intermediate clients. The results are identical to the small clients. Thus, there is a clear difference in performance for the learning rate value of 0.1 from the rest of the values, which achieve similar results with 0.001 being able to achieve better results than the other values. For the FedAvg (Figure 5.13(b)) algorithm, all the learning rates achieve similar results, including the value 0.1, while for LG-FedAvg (Figure 5.14(b)), the gap between the value 0.1 and the remaining learning rates is also present and the value 0.001 achieves the best results.

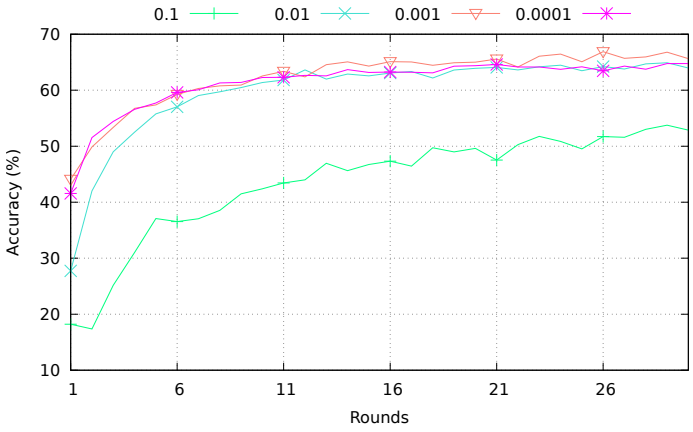
Performance for the Big Clients Figure 5.12(b) shows the performance of the *FedHybridAvgLGDual* algorithm for different learning rate values for the big clients. The results are similar to the previous clients, hence the learning rate 0.1 achieves the worse results with 0.001 achieving the best performance and the remaining two values being close to each other in performance. Similar results can be observed for the FedAvg and LG-FedAvg algorithms (Figures 5.13(c) and 5.14(c), respectively).

Discussion of Results From the obtained results, we can see that the highest value of the learning rate tested, 0.1, is the worse performing for each type of client. The higher the learning rate, the more heterogeneous the client updates are, as such, it is expected for this value to have the worse performance, since the client updates are more personalized to their data each round, resulting in an aggregated model which is less generalizable. If we look at the results of the smaller clients, for our hybrid approach and for FedAvg, we can clearly see the influence of the aggregation of the more specialized heads of the larger clients which results in a more specialized global head, since in our approach the 0.1 learning rate value has a much bigger gap in performance to the other values as the global head is more specialized, while that gap is way smaller for FedAvg with a more general head.

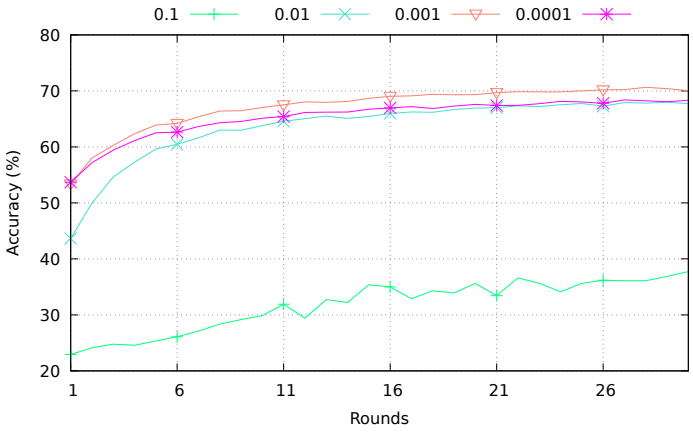
For all the types of clients, the learning rate value of 0.001 is the best-performing. Therefore, this value can achieve the best compromise between allowing the larger clients to personalize their models sufficiently (the lower the learning rate, the less each training step modifies the model, thus the less the personalization) and their local updates not being too heterogeneous (the higher the learning rate, the more each training step modifies the model, thus the more the local update heterogeneity, resulting in a global model which is not general and, as such, not fit for the generality of the clients).



(a) Small Clients

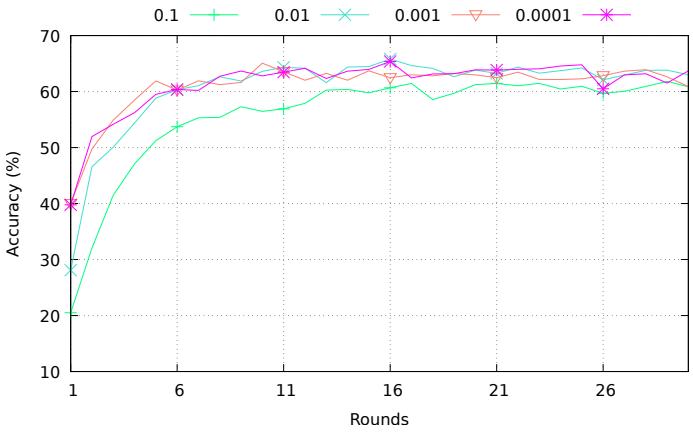


(b) Intermediate Clients

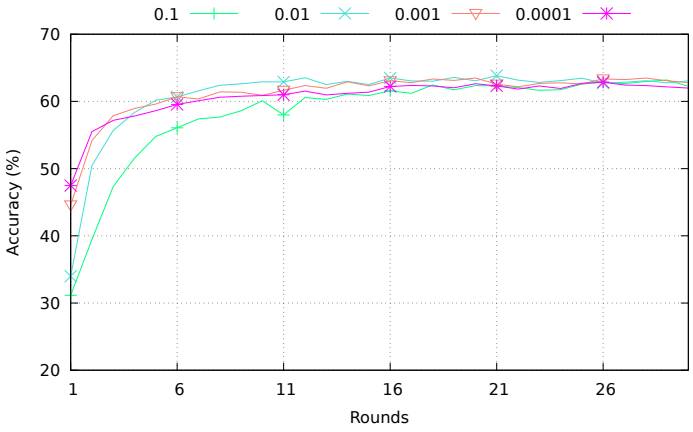


(c) Big Clients

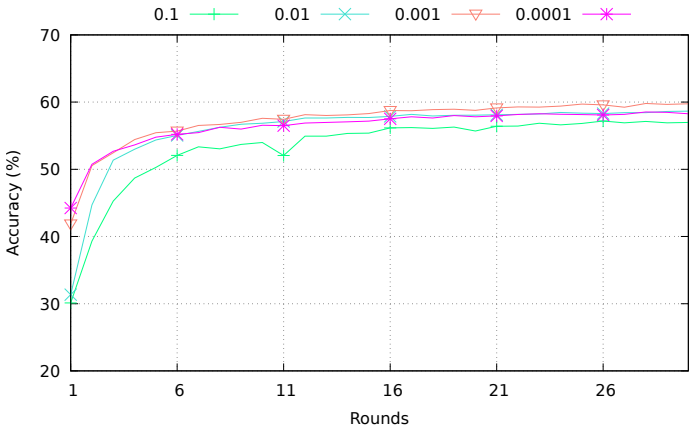
Figure 5.12: Accuracy of the various models for clients of different sizes for the *FedHybridAvgLGDual* when varying the learning rate.



(a) Small Clients

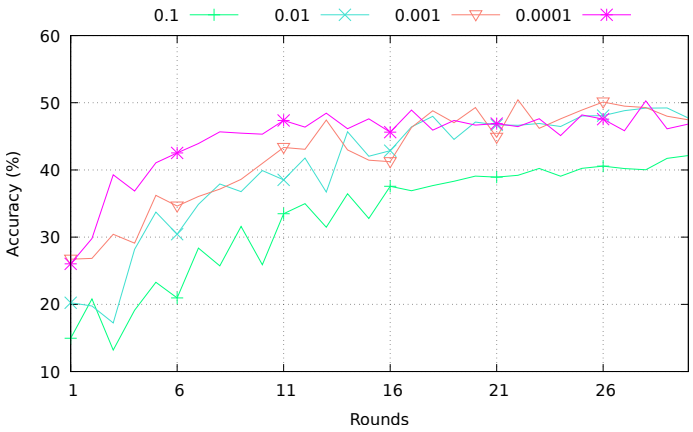


(b) Intermediate Clients

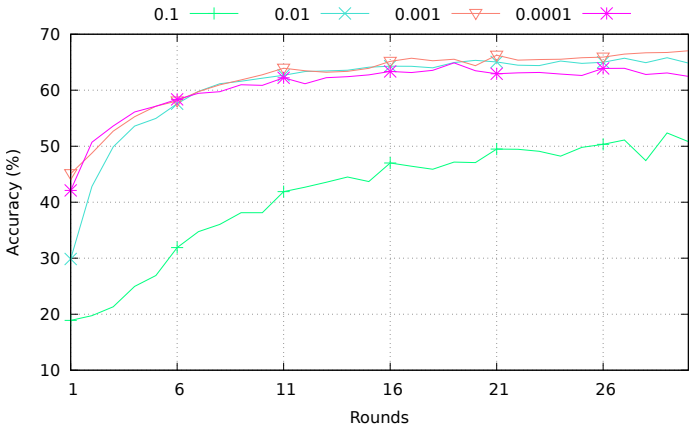


(c) Big Clients

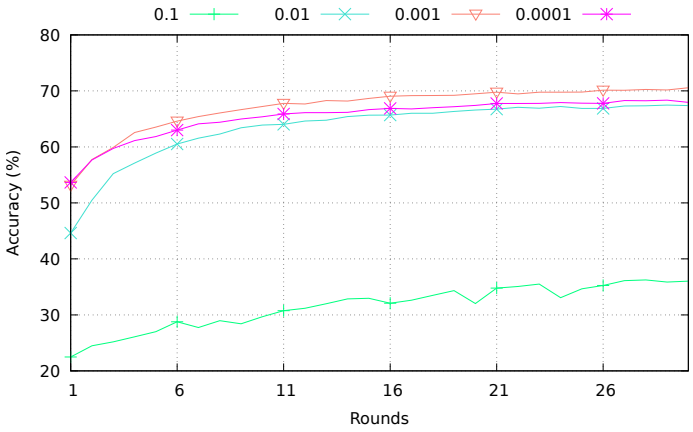
Figure 5.13: Accuracy of the various models for clients of different sizes for the FedAvg algorithm when varying the learning rate.



(a) Small Clients



(b) Intermediate Clients



(c) Big Clients

Figure 5.14: Accuracy of the various models for clients of different sizes for the LG-FedAvg algorithm when varying the learning rate.

Summary

This chapter has described an experimental study of some of the different algorithms for personalized FL mentioned in the literature and of our hybrid proposals. From the results, we have observed that smaller clients achieve better performance when the full model is exchanged with the server, whereas clients with more data achieve better results when only the head is exchanged and the body is personalized. From the two hybrid proposals, only *FedHybridAvgLGDual* performed as intended and managed to perform similarly to the top algorithms for small, intermediate and big clients. We have also shown that while fine-tuning improves some of the literature algorithms' performance, it has a much less considerable impact on the *FedHybridAvgLGDual* algorithm. Also, we have shown that for a more complex task our hybrid proposal still performs similarly to the top algorithms in almost all client classes, except for the intermediate clients where FedRep manages to achieve slightly better performance. We have observed that our proposal is not able to suggest newly released actions, a problem that seems to be common among most of the personalized algorithms studied. We have also shown that there is a trade-off when performing more local computation rounds in the *FedHybridAvgLGDual* algorithms, since smaller clients achieve less performance with more local rounds, while bigger clients achieve better performance. Finally, we have shown that the learning rate can influence the performance of the algorithms, higher learning rates achieve lower performance, due to the more heterogeneous updates, however, learning rates that are too low also do not achieve the best performance since the personalization is lower.

6

Conclusion

Contents

6.1 Conclusions

Federated Learning is an Machine Learning approach that allows clients to collaboratively train a global model with their own private data without its privacy being compromised. In this thesis, we performed an experimental study to evaluate the viability of applying techniques of personalized FL to our use case, the *Service Studio* platform developed by OUTSYSTEMS. In this platform, GNNs are used in order to recommend possible next actions that the users might want to add to an action flow.

We surveyed some solutions proposed in the literature and evaluated them to assess the possibility of substituting the current centralized model for federated algorithms which allow the creation of personalized models for each client. The obtained results demonstrated that the amount of data of each client influences the performance of each algorithm, meaning there is no algorithm which works well for every client. Clients with fewer data prefer an algorithm which allows collaboration on the full model, as they do not have enough data to personalize part of the model. Clients with more data, prefer to collaborate on the head of the model and personalize the body. Hence, we also proposed and evaluated possible approaches that merge some of the studied algorithms, which we call hybrid algorithms. One of the proposed algorithms, *FedHybridAvgLGDual*, which merges the FedAvg and LG-FedAvg algorithms, proved to achieve similar performance to the top algorithms for all the types of clients for the typical OUTSYSTEMS ML task, which is to predict the kind of the next action to be added to the action flow. We also demonstrated that for this task fine-tuning only improves the performance for clients with a low amount of data in our hybrid proposal, which is not the case for most of the literature algorithms. Also, we tested this algorithm with a more complex task with more output classes and demonstrated that it still performs similarly to the top algorithms in all but one class of clients where one of the literature algorithms achieves slightly better performance. Furthermore, the increase in local computation rounds improves the performance of the model for clients with more data at the expense of loss of performance for clients with fewer data. Moreover, the learning rate can influence the performance of the models obtained, since higher learning rates achieve lower performance, due to the more heterogeneous updates, and learning rates that are too low also do not achieve the best performance as the personalization is lower. Lastly, we have shown that our hybrid algorithm is not able to suggest newly released actions, a pattern that is also observed in most of the algorithms studied in the literature.

6.2 Limitations and Future Work

Our proposed algorithm *FedHybridAvgLGDual* has some limitations. Firstly, it requires the clients with more data (intermediate and big clients) to calculate two different models in each communication round. Secondly, as seen in one of the experiments performed, due to having one of the parts of the model specialized, the model for larger clients does not seem to be able to recommend new actions introduced

to the platform.

Therefore, as future work, other hybrid approaches should be explored in order to avoid having to calculate two different models for larger clients. Also, a mechanism to "recalibrate" the models whenever a new action is introduced is also needed, perhaps by averaging the global body with the local body of the larger clients, so that the body learns how to represent the novel action. Furthermore, as only parameter decoupling techniques were analysed, other FL personalization approaches could be tested, for instance, meta-learning or clustering. Moreover, security and privacy defence mechanisms need to be implemented as we only focused on the personalization challenge of FL. The number of experiments we have been able to perform was constrained by the budget available to spend on cloud resources and by the duration of the academic year. Naturally, more tests could be executed: tests with other data sets (possibly even out of the scope of OUTSYSTEMS) and; tests to understand the influence of varying the hyperparameters, for instance, testing more values of the number of local training epochs or testing unexplored hyperparameters like the fraction of selected clients in every communication round. Lastly, as discussed in one of the experiments, it would also be interesting to perform further tests to understand how FedBABU manages to recommend new actions before and after fine-tuning.

Bibliography

[1] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, October 2018. [Online]. Available: http://arxiv.org/abs/1806.01261

[2] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, P. P. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," *CoRR*, vol. abs/2007.14390, July 2020. [Online]. Available: https://arxiv.org/abs/2007.14390

[3] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," in *Proceedings of Machine Learning and Systems 2019, MLSys 2019*. Stanford, CA, USA: mlsys.org, March 2019, pp. 374–388. [Online]. Available: https://proceedings.mlsys.org/book/271.pdf

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, ser. Proceedings of Machine Learning Research, vol. 54. Fort Lauderdale, FL, USA: PMLR, April 2017, pp. 1273–1282. [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a.html

[5] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, June 2017. [Online]. Available: http://arxiv.org/abs/1609.04747

[6] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, pp. 1–210, June 2021. [Online]. Available: https://doi.org/10.1561/2200000083

[7] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: vision, hype and reality for data privacy and protection," *IEEE*

Transactions on Knowledge and Data Engineering, November 2021. [Online]. Available: https://doi.org/10.1109/TKDE.2021.3124599

[8] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 12:1–12:19, January 2019. [Online]. Available: https://doi.org/10.1145/3298981

[9] A. Blanco-Justicia, J. Domingo-Ferrer, S. Martínez, D. Sánchez, A. Flanagan, and K. E. Tan, "Achieving security and privacy in federated learning systems: Survey, research challenges and future directions," *Engineering Applications of Artificial Intelligence*, vol. 106, p. 104468, November 2021. [Online]. Available: https://doi.org/10.1016/j.engappai.2021.104468

[10] L. Lyu, H. Yu, X. Ma, L. Sun, J. Zhao, Q. Yang, and P. S. Yu, "Privacy and robustness in federated learning: Attacks and defenses," *CoRR*, vol. abs/2012.06337, December 2020. [Online]. Available: https://arxiv.org/abs/2012.06337

[11] X. Yin, Y. Zhu, and J. Hu, "A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 131:1–131:36, July 2021. [Online]. Available: https://doi.org/10.1145/3460427

[12] N. Truong, K. Sun, S. Wang, F. Guitton, and Y. Guo, "Privacy preservation in federated learning: An insightful survey from the gdpr perspective," *Computers and Security*, vol. 110, p. 102402, November 2021. [Online]. Available: https://doi.org/10.1016/j.cose.2021.102402

[13] Y. Sun, H. Ochiai, and H. Esaki, "Decentralized deep learning for multi-access edge computing: A survey on communication efficiency and trustworthiness," *IEEE Transactions on Artificial Intelligence*, vol. 1, December 2021. [Online]. Available: https://doi.org/10.1109/TAI.2021.3133819

[14] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning," in *2022 IEEE Symposium on Security and Privacy, , SP 2022*. San Francisco, CA, USA: IEEE, May 2022, pp. 1354–1371. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833647

[15] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards personalized federated learning," *CoRR*, vol. abs/2103.00710, March 2021. [Online]. Available: https://arxiv.org/abs/2103.00710

[16] M. Jiang, T. Jung, R. Karl, and T. Zhao, "Federated dynamic gnn with secure aggregation," *CoRR*, vol. abs/2009.07351, September 2020. [Online]. Available: https://arxiv.org/abs/2009.07351

[17] C. Chen, J. Zhou, L. Zheng, H. Wu, L. Lyu, J. Wu, B. Wu, Z. Liu, L. Wang, and X. Zheng, "Vertically federated graph neural network for privacy-preserving node classification," in *Proceedings of the*

Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022. Vienna, Austria: ijcai.org, July 2022, pp. 1959–1965. [Online]. Available: https://doi.org/10.24963/ijcai.2022/272

[18] B. Wang, A. Li, H. Li, and Y. Chen, "Graphfl: A federated learning framework for semi-supervised node classification on graphs," *CoRR*, vol. abs/2012.04187, December 2020. [Online]. Available: https://arxiv.org/abs/2012.04187

[19] H. Xie, J. Ma, L. Xiong, and C. Yang, "Federated graph classification over non-iid graphs," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*. virtual: Curran Associates, Inc., December 2021, pp. 18 839–18 852. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/9c6947bd95ae487c81d4e19d3ed8cd6f-Abstract.html

[20] K. Zhang, C. Yang, X. Li, L. Sun, and S. M. Yiu, "Subgraph federated learning with missing neighbor generation," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*. virtual: Curran Associates, Inc., December 2021, pp. 6671–6682. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/34adeb8e3242824038aa65460a47c29e-Abstract.html

[21] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang, "Fate: An industrial grade platform for collaborative learning with data protection," *Journal of Machine Learning Research*, vol. 22, no. 226, pp. 1–6, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-815.html

[22] C. He, K. Balasubramanian, E. Ceyani, C. Yang, H. Xie, L. Sun, L. He, L. Yang, P. S. Yu, Y. Rong *et al.*, "Fedgraphnn: A federated learning system and benchmark for graph neural networks," *CoRR*, vol. abs/2104.07145, April 2021. [Online]. Available: https://arxiv.org/abs/2104.07145

[23] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu *et al.*, "Fedml: A research library and benchmark for federated machine learning," *CoRR*, vol. abs/2007.13518, July 2020. [Online]. Available: https://arxiv.org/abs/2007.13518

[24] J. Konečnỳ, 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, October 2017. [Online]. Available: http://arxiv.org/abs/1610.05492

[25] S. Caldas, J. Konečny, H. B. McMahan, and A. Talwalkar, "Expanding the reach of federated learning by reducing client resource requirements," *CoRR*, vol. abs/1812.07210, January 2019. [Online]. Available: http://arxiv.org/abs/1812.07210

[26] W. Luping, W. Wei, and L. Bo, "Cmfl: Mitigating communication overhead for federated learning," in *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019*. Dallas, TX, USA: IEEE, July 2019, pp. 954–964. [Online]. Available: https://doi.org/10.1109/ICDCS.2019.00099

[27] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *2019 IEEE Conference on Computer Communications, INFOCOM 2019*. Paris, France: IEEE, April 2019, pp. 2512–2520. [Online]. Available: https://doi.org/10.1109/INFOCOM.2019.8737416

[28] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE Symposium on Security and Privacy, SP 2019*. San Francisco, CA, USA: IEEE, May 2019, pp. 739–753. [Online]. Available: https://doi.org/10.1109/SP.2019.00065

[29] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE Symposium on Security and Privacy, SP 2019*. San Francisco, CA, USA: IEEE, May 2019, pp. 691–706. [Online]. Available: https://doi.org/10.1109/SP.2019.00029

[30] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*. Vancouver, BC, Canada: Curran Associates, Inc., December 2019, pp. 14 747–14 756. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/60a6c4002cc7b29142def8871531281a-Abstract.html

[31] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas, TX, USA: ACM, October 2017, pp. 1175–1191. [Online]. Available: https://doi.org/10.1145/3133956.3133982

[32] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, May 2018. [Online]. Available: https://doi.org/10.1109/TIFS.2017.2787987

[33] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *CoRR*, vol. abs/1712.07557, March 2018. [Online]. Available: http://arxiv.org/abs/1712.07557

[34] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, “Learning differentially private recurrent language models,” in *6th International Conference on Learning Representations, ICLR 2018*. Vancouver, BC, Canada: OpenReview.net, April 2018. [Online]. Available: https://openreview.net/forum?id=BJ0hF1Z0b

[35] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, “Data poisoning attacks against federated learning systems,” in *Computer Security – ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020*, ser. Lecture Notes in Computer Science, vol. 12308. Guildford, UK: Springer, September 2020, pp. 480–501. [Online]. Available: https://doi.org/10.1007/978-3-030-58951-6_24

[36] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” in *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 108. Virtual Event: PMLR, August 2020, pp. 2938–2948. [Online]. Available: http://proceedings.mlr.press/v108/bagdasaryan20a.html

[37] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent,” in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, NeurIPS 2017*. Long Beach, CA, USA: Curran Associates, Inc., December 2017, pp. 119–129. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/f4b9ec30ad9f68f89b29639786cb62ef-Abstract.html

[38] C. Fung, C. J. Yoon, and I. Beschastnikh, “Mitigating sybils in federated learning poisoning,” *CoRR*, vol. abs/1808.04866, July 2020. [Online]. Available: http://arxiv.org/abs/1808.04866

[39] A. Fallah, A. Mokhtari, and A. Ozdaglar, “Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach,” in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*. Virtual Event: Curran Associates, Inc., December 2020, pp. 3557–3568. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/24389bfe4fe2eba8bf9aa9203a44cdad-Abstract.html

[40] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, “Exploiting shared representations for personalized federated learning,” in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, ser. Proceedings of Machine Learning Research, vol. 139. Virtual Event: PMLR, July 2021, pp. 2089–2099. [Online]. Available: http://proceedings.mlr.press/v139/collins21a.html

[41] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” in *Proceedings of Machine Learning and Systems 2020, MLSys*

2020, vol. 2. Austin, TX, USA: mlsys.org, March 2020, pp. 429–450. [Online]. Available: https://proceedings.mlsys.org/book/316.pdf

[42] C. T. Dinh, T. T. Vu, N. H. Tran, M. N. Dao, and H. Zhang, "A new look and convergence rate of federated multi-task learning with laplacian regularization," *CoRR*, vol. abs/2102.07148, October 2022. [Online]. Available: https://arxiv.org/abs/2102.07148

[43] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," *CoRR*, vol. abs/1912.00818, December 2019. [Online]. Available: http://arxiv.org/abs/1912.00818

[44] P. P. Liang, T. Liu, L. Ziyin, N. B. Allen, R. P. Auerbach, D. Brent, R. Salakhutdinov, and L.-P. Morency, "Think locally, act globally: Federated learning with local and global representations," *CoRR*, vol. abs/2001.01523, July 2020. [Online]. Available: http://arxiv.org/abs/2001.01523

[45] J. Oh, S. Kim, and S.-Y. Yun, "Fedbabu: Towards enhanced representation for federated image classification," *CoRR*, vol. abs/2106.06042, June 2021. [Online]. Available: https://arxiv.org/abs/2106.06042

[46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*. Vancouver, BC, Canada: Curran Associates, Inc., December 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

A

Hybrid Algorithms Pseudocode

Notation:

- C represents the set of total clients;
- f represents the fraction of clients that participate in every communication round;
- w_G^t represents the global model parameters calculated in round t which are divided into: $w_{G,body}^t$ the body parameters; and $w_{G,head}^t$ the head parameters;
- w_i^t represents the local update for client with index i of set S^t;
- n_i^t represents the data set size for client with index i of set S^t;
- E represents the number of local computation epochs;
- $w_{k,body}$ represents the local body for client k and $w_{k,head}$ the local head;
- SGD$(w_b, w_h, \mathcal{D}_k, B)$ means executing one step of the SGD algorithm for each batch of size B from the data set of client k, $\mathcal{D}_k$, for a model with body parameters w_b and head parameters w_h;

- $size(\mathcal{D})$ indicates the size of the data set $\mathcal{D}$.
- $threshold$ indicates the constant defined for considering a client small.

Algorithm 1 Training Procedure for the *FedHybridAvgLG* algorithm.

initialize $w_G^0 = \{w_{G,body}^0 , w_{G,head}^0\}$ ▷ Server Executes
for each $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot f, 1)$
  $S^t \leftarrow$ random set of m clients from C
  for each $k \in \mathcal{S}^t$ in parallel **do**
    $w_k^t \leftarrow$ ClientLocalUpdate(w_G^{t-1})
  $n \leftarrow \sum_{i=1}^{m} n_i^t$
  $w_G^t \leftarrow \sum_{i=1}^{m} \frac{n_i^t}{n} w_i^t$

procedure CLIENTLOCALUPDATE(w_G^{t-1}) ▷ Client executes
  $\{w_{G,body}^{t-1}, w_{G,head}^{t-1}\} \leftarrow w_G^{t-1}$
  $small\_client \leftarrow isSmallClient(\mathcal{D}_k)$
  if $small\_client$ **then**
    $\{w_{k,body}, w_{k,head}\} \leftarrow \{w_{G,body}^{t-1}, w_{G,head}^{t-1}\}$
  else
    $w_{k,head} \leftarrow w_{G,head}^{t-1}$ ▷ $w_{k,body}$ is maintained locally for bigger clients
  for each $1, 2, \dots, E$ **do**
    $\{w_{k,body}, w_{k,head}\} \leftarrow SGD(w_{k,body}, w_{k,head}, \mathcal{D}_k, B)$
  $w_k \leftarrow \{w_{k,body}, w_{k,head}\}$
  returns w_k

procedure ISSMALLCLIENT($\mathcal{D}_k$) ▷ Client executes
  if $size(\mathcal{D}_k) > threshold$ **then**
    returns $False$
  else returns $True$

Algorithm 2 Training Procedure for the *FedHybridAvgLGDual* algorithm.

initialize $w_G^0 = \{w_{G,body}^0 , w_{G,head}^0\}$ ▷ Server Executes
for each $t = 1, 2, \dots$ **do**
    $m \leftarrow \max(C \cdot f, 1)$
    $S^t \leftarrow$ random set of m clients from C
    for each $k \in \mathcal{S}^t$ in parallel **do**
        $w_k^t \leftarrow$ ClientLocalUpdate(w_G^{t-1})
    $n \leftarrow \sum_{i=1}^{m} n_i^t$
    $w_G^t \leftarrow \sum_{i=1}^{m} \frac{n_i^t}{n} w_i^t$

procedure CLIENTLOCALUPDATE(w_G^{t-1}) ▷ Client executes
    $\{w_{G,body}^{t-1}, w_{G,head}^{t-1}\} \leftarrow w_G^{t-1}$
    $small\_client \leftarrow isSmallClient(\mathcal{D}_k)$
    if $small\_client$ **then**
        $\{w_{k,body}, w_{k,head}\} \leftarrow \{w_{G,body}^{t-1}, w_{G,head}^{t-1}\}$
        for each $1, 2, \dots, E$ **do** ▷ FedAvg training
            $\{w_{k,body}, w_{k,head}\} \leftarrow SGD(w_{k,body}, w_{k,head}, \mathcal{D}_k, B)$
        $w_k \leftarrow \{w_{k,body}, w_{k,head}\}$
    else
        $\{w_{k,body\_avg}, w_{k,head}\} \leftarrow \{w_{G,body}^{t-1}, w_{G,head}^{t-1}\}$
        for each $1, 2, \dots, E$ **do** ▷ FedAvg training
            $\{w_{k,body\_avg}, w_{k,head}\} \leftarrow SGD(w_{k,body\_avg}, w_{k,head}, \mathcal{D}_k, B)$
        $w_{k,head} \leftarrow w_{G,head}^{t-1}$ ▷ $w_{k,body}$ is maintained locally
        for each $1, 2, \dots, E$ **do** ▷ LG-FedAvg training
            $\{w_{k,body}, w_{k,head}\} \leftarrow SGD(w_{k,body}, w_{k,head}, \mathcal{D}_k, B)$
        $w_k \leftarrow \{w_{k,body\_avg}, w_{k,head}\}$
    returns w_k

procedure ISSMALLCLIENT($\mathcal{D}_k$) ▷ Client executes
    if $size(\mathcal{D}_k) > threshold$ **then**
        returns $False$
    else returns $True$

B

Class Distribution

B.1 Node Kind Task Class Distribution

Table B.1: Class distribution of the 33 selected clients for the node kind task.

Class Name	Mean (%)	Std-dev (%)
Nodes.AjaxRefresh	2.86	2.92
Nodes.Assign	18.33	4.56
Nodes.AttachEmailContent	0.01	0.03
Nodes.Comment	0.00	0.00
Nodes.DataSet	5.58	2.66
Nodes.Download	0.24	0.24
Nodes.End	22.93	3.18
Nodes.ErrorHandler	0.00	0.00
Nodes.ExcelToRecordList	0.28	0.46
Nodes.ExecuteAction	15.65	5.27
Nodes.ForEach	2.07	1.04
Nodes.If	10.77	2.28
Nodes.JSONDeserialize	0.13	0.18
Nodes.JSONSerialize	0.23	0.36
Nodes.Outcome	0.14	0.30
Nodes.RaiseError	0.77	1.36
Nodes.RecordListToExcel	0.13	0.16
Nodes.RefreshQuery	2.29	1.59
Nodes.SendEmail	0.10	0.15
Nodes.Start	0.00	0.00
Nodes.Switch	0.38	0.37
Nodes.WebDestination	3.64	2.86
NRNodes.ExecuteClientAction	6.14	7.02
NRNodes.FeedbackMessage	5.60	2.74
NRNodes.JavascriptNode	0.98	1.15
NRNodes.TriggerEvent	0.75	0.93

B.2 Node Subkind Task Class Distribution

Table B.2: Class distribution of the 33 selected clients for the node subkind task.

Class Name	Mean (%)	Std-dev (%)
Nodes.AjaxRefresh	2.86	2.92
Nodes.Assign	18.33	4.56
Nodes.AttachEmailContent	0.01	0.03
Nodes.Comment	0.00	0.00
Nodes.DataSet	5.58	2.66
Nodes.Download	0.24	0.24
Nodes.End	22.93	3.18
Nodes.ErrorHandler	0.00	0.00
Nodes.ExcelToRecordList	0.28	0.46
Nodes.ExecuteAction	8.66	3.92
Nodes.ExecuteAction.EntityActions.CreateEntity	0.67	0.69
Nodes.ExecuteAction.EntityActions.CreateOrUpdateAllEntity	0.00	0.00
Nodes.ExecuteAction.EntityActions.CreateOrUpdateEntity	0.74	0.66
Nodes.ExecuteAction.EntityActions.DeleteEntity	0.44	0.40
Nodes.ExecuteAction.EntityActions.GetEntity	0.10	0.15
Nodes.ExecuteAction.EntityActions.GetEntityForUpdate	0.09	0.14
Nodes.ExecuteAction.EntityActions.UpdateEntity	0.41	0.41
Nodes.ExecuteAction.NRFlows.ClientActionFlow	0.00	0.00
Nodes.ExecuteAction.NRFlows.ClientScreenActionFlow	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.(System).AbortTransaction	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.(System).ActivityClose	0.01	0.02
Nodes.ExecuteAction.ReferenceAction.(System).CommitTransaction	0.09	0.13
Nodes.ExecuteAction.ReferenceAction.(System).EspaceInvalidateCache	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.(System).GenerateGuid	0.01	0.02

Table B.2: Class distribution of the 33 selected clients for the node subkind task.

Class Name	Mean (%)	Std-dev (%)
Nodes.ExecuteAction.ReferenceAction.(System).ListAll	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.(System).ListAny	0.05	0.12
Nodes.ExecuteAction.ReferenceAction.(System).ListAppend	0.85	0.90
Nodes.ExecuteAction.ReferenceAction.(System).ListAppendAll	0.23	0.26
Nodes.ExecuteAction.ReferenceAction.(System).ListClear	0.27	0.34
Nodes.ExecuteAction.ReferenceAction.(System).ListDistinct	0.01	0.02
Nodes.ExecuteAction.ReferenceAction.(System).ListDuplicate	0.01	0.01
Nodes.ExecuteAction.ReferenceAction.(System).ListFilter	0.29	0.34
Nodes.ExecuteAction.ReferenceAction.(System).ListIndexOf	0.09	0.12
Nodes.ExecuteAction.ReferenceAction.(System).ListInsert	0.05	0.17
Nodes.ExecuteAction.ReferenceAction.(System).ListRemove	0.09	0.18
Nodes.ExecuteAction.ReferenceAction.(System).ListSort	0.08	0.15
Nodes.ExecuteAction.ReferenceAction.(System).Login	0.03	0.06
Nodes.ExecuteAction.ReferenceAction.(System).LogMessage	0.21	0.27
Nodes.ExecuteAction.ReferenceAction.(System).Logout	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.(System).SetCurrentLocale	0.03	0.07
Nodes.ExecuteAction.ReferenceAction.(System).TenantSwitch	0.01	0.04
Nodes.ExecuteAction.ReferenceAction.ardoHTTP.HTTPGet	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.ardoHTTP.HTTPPost	0.01	0.06
Nodes.ExecuteAction.ReferenceAction.ardoJSON.JSONSelect	0.02	0.07
Nodes.ExecuteAction.ReferenceAction.AsynchronousLogging.LogError	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.Audit.AuditUI_OperationEnd	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.Audit.AuditUI_OperationErrorAndEnd	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.AuditAdmin.AddAudit	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.BinaryData.Base64ToBinary	0.01	0.04
Nodes.ExecuteAction.ReferenceAction.BinaryData.BinaryDataSize	0.01	0.03
Nodes.ExecuteAction.ReferenceAction.BinaryData.BinaryDataToText	0.01	0.02
Nodes.ExecuteAction.ReferenceAction.BinaryData.BinaryToBase64	0.01	0.01
Nodes.ExecuteAction.ReferenceAction.BinaryData.TextToBinaryData	0.03	0.06
Nodes.ExecuteAction.ReferenceAction.Charts.AdvancedFormat_Init	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.Charts.DataPoint_Init	0.01	0.02
Nodes.ExecuteAction.ReferenceAction.HashTable.add	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.HashTable.get	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.HtmlToPdfConverter.GeneratePDF	0.02	0.07
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.AddHeader	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.AddJavaScriptTag	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.AddLinkTag	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.AddMetaTag	0.01	0.01
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.GetCookie	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.GetEntryURL	0.01	0.02
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.GetRequestHeader	0.02	0.07
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.GetURL	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.SetCookie	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.SetRequestTimeout	0.01	0.02
Nodes.ExecuteAction.ReferenceAction.HTTPRequestHandler.SetStatusCode	0.01	0.03
Nodes.ExecuteAction.ReferenceAction.OfficeUtils.Excel_Export_GenerateFile	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.PlatformPasswordUtils.ValidatePassword	0.01	0.04
Nodes.ExecuteAction.ReferenceAction.RandomizerNumberGenerator.GetRandomInt	0.00	0.00

Table B.2: Class distribution of the 33 selected clients for the node subkind task.

Class Name	Mean (%)	Std-dev (%)
Nodes.ExecuteAction.ReferenceAction.RichWidgets.Input_AutoComplete_GetIdentifier	0.02	0.06
Nodes.ExecuteAction.ReferenceAction.RichWidgets.Input_FocusFirstInvalid	0.13	0.17
Nodes.ExecuteAction.ReferenceAction.RichWidgets.List_Navigation_ResetStartIndex	0.34	0.40
Nodes.ExecuteAction.ReferenceAction.RichWidgets.Popup_Editor_Close	0.17	0.27
Nodes.ExecuteAction.ReferenceAction.RichWidgets.Popup_Editor_Notify	0.10	0.15
Nodes.ExecuteAction.ReferenceAction.SortRecordList.SortRecordList	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.Text.Regex_Replace	0.01	0.03
Nodes.ExecuteAction.ReferenceAction.Text.Regex_Search	0.05	0.06
Nodes.ExecuteAction.ReferenceAction.Text.String_Join	0.04	0.07
Nodes.ExecuteAction.ReferenceAction.Text.String_Split	0.09	0.14
Nodes.ExecuteAction.ReferenceAction.Text.StringBuilder_Append	0.01	0.03
Nodes.ExecuteAction.ReferenceAction.Text.StringBuilder_Create	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.Text.StringBuilder_ToString	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.Users.EncryptPassword	0.01	0.02
Nodes.ExecuteAction.ReferenceAction.Users.User_GetUnifiedLoginUrl	0.42	0.55
Nodes.ExecuteAction.ReferenceAction.Users.User_Login	0.24	0.26
Nodes.ExecuteAction.ReferenceAction.Users.User_Logout	0.25	0.26
Nodes.ExecuteAction.ReferenceAction.Xml.XmlDocument_Load	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.Xml.XmlElement_AppendChildElement	0.01	0.06
Nodes.ExecuteAction.ReferenceAction.Xml.XmlElement_GetAttributeValue	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.Xml.XmlElement_GetInnerText	0.00	0.00
Nodes.ExecuteAction.ReferenceAction.Xml.XmlElement_SelectSingleNode	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.Xml.XmlNodeList_Count	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.Xml.XmlNodeList_Item	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.Zip.AddFile	0.01	0.02
Nodes.ExecuteAction.ReferenceAction.Zip.CommitChanges	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.Zip.CreateZip	0.00	0.01
Nodes.ExecuteAction.ReferenceAction.Zip.GetZipBinary	0.00	0.01
Nodes.ExecuteAction.SystemActions.OnBeginWebRequest	0.00	0.00
Nodes.ExecuteAction.SystemActions.OnSessionStart	0.00	0.00
Nodes.ForEach	2.07	1.04
Nodes.If	0.00	0.00
Nodes.If.Empty	1.07	0.55
Nodes.If.FailedToParse	0.00	0.00
Nodes.If.None	0.00	0.00
Nodes.If.NotEmpty	0.19	0.37
Nodes.If.NotOthers	0.01	0.01
Nodes.If.NotValid	0.00	0.01
Nodes.If.NotZeroNullOrEmpty	1.58	1.03
Nodes.If.Others	6.61	2.30
Nodes.If.Valid	0.39	0.42
Nodes.If.ZeroNullOrEmpty	0.93	0.74
Nodes.JSONDeserialize	0.13	0.18
Nodes.JSONSerialize	0.23	0.36
Nodes.Outcome	0.14	0.30
Nodes.RaiseError	0.77	1.36
Nodes.RecordListToExcel	0.13	0.16
Nodes.RefreshQuery	2.29	1.59

Table B.2: Class distribution of the 33 selected clients for the node subkind task.

Class Name	Mean (%)	Std-dev (%)
Nodes.SendEmail	0.10	0.15
Nodes.Start	0.00	0.00
Nodes.Switch	0.38	0.37
Nodes.WebDestination	3.64	2.86
NRNodes.ExecuteClientAction	4.58	5.48
NRNodes.ExecuteClientAction.EntityActions.CreateEntity	0.03	0.06
NRNodes.ExecuteClientAction.EntityActions.CreateOrUpdateAllEntity	0.06	0.15
NRNodes.ExecuteClientAction.EntityActions.CreateOrUpdateEntity	0.11	0.33
NRNodes.ExecuteClientAction.EntityActions.DeleteAllEntity	0.13	0.41
NRNodes.ExecuteClientAction.EntityActions.DeleteEntity	0.04	0.11
NRNodes.ExecuteClientAction.EntityActions.GetEntity	0.02	0.08
NRNodes.ExecuteClientAction.EntityActions.UpdateEntity	0.03	0.07
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).ListAny	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).ListAppend	0.43	0.84
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).ListAppendAll	0.09	0.23
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).ListClear	0.20	0.43
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).ListDuplicate	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).ListFilter	0.13	0.28
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).ListIndexOf	0.06	0.16
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).ListInsert	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).ListRemove	0.07	0.21
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).ListSort	0.05	0.14
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).LogMessage	0.01	0.03
NRNodes.ExecuteClientAction.ReferenceClientAction.(System).RequireScript	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.BarcodePlugin.CheckBarcodePlugin	0.00	0.02
NRNodes.ExecuteClientAction.ReferenceClientAction.BarcodePlugin.ScanBarcode	0.00	0.02
NRNodes.ExecuteClientAction.ReferenceClientAction.CameraPlugin.CheckCameraPlugin	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.CameraPlugin.TakeFromPhone	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.CameraPlugin.TakePicture	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.CommonPlugin.ConsoleLog	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.CommonPlugin.GetDeviceID	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.CommonPlugin.GetOperatingSystem	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.CommonPlugin.GetPlatform	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.CommonPlugin.IsCordovaDefined	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.DevicePlugin.CheckDevicePlugin	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.DevicePlugin.GetDeviceInfo	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.FilePlugin.CheckFilePlugin	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.FilePlugin.GetFileDataFromUri	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.FilePlugin.SaveFile	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.FileViewerPlugin.CheckDocumentViewerPlugin	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.FileViewerPlugin.OpenDocument	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.FirebaseMobile.LogEvent	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.InAppBrowserPlugin.CheckInAppBrowserPlugin	0.01	0.06
NRNodes.ExecuteClientAction.ReferenceClientAction.InAppBrowserPlugin.Open	0.02	0.10
NRNodes.ExecuteClientAction.ReferenceClientAction.InputMasksMobile.ForceUpdate	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.KeyStorePlugin.CheckKeyStorePlugin	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.KeyStorePlugin.GetValue	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.KeyStorePlugin.RemoveKey	0.00	0.00

Table B.2: Class distribution of the 33 selected clients for the node subkind task.

Class Name	Mean (%)	Std-dev (%)
NRNodes.ExecuteClientAction.ReferenceClientAction.KeyStorePlugin.SetValue	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.LocationPlugin.CheckLocationPlugin	0.00	0.02
NRNodes.ExecuteClientAction.ReferenceClientAction.LocationPlugin.GetLocation	0.00	0.02
NRNodes.ExecuteClientAction.ReferenceClientAction.MobilePatterns.ConfigureOfflineDataSync	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.MobilePatterns.EndOfflineDataSync	0.00	0.02
NRNodes.ExecuteClientAction.ReferenceClientAction.MobilePatterns.GetNetworkStatus	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.MobilePatterns.LayoutReady	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.MobilePatterns.MenuHide	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.MobilePatterns.MenuShow	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.MobilePatterns.SetMenuIcon	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.MobilePatterns.ShowPassword	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.MobilePatterns.StartOfflineDataSync	0.00	0.01
NRNodes.ExecuteClientAction.ReferenceClientAction.MultiLingual.AddTranslationsFromResource	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.MultiLingual.GetLocale	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.MultiLingual.SetLocale	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.OneSignalPlugin.CheckOneSignalPlugin	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.OneSignalPlugin.Register	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.OneSignalPlugin.RegisterWithUser	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.OneSignalPlugin.SetTag	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.PushwooshPlugin.CheckPushwooshPlugin	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.PushwooshPlugin.RegisterDevice	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.ScreenOrientationPlugin.LockOrientation	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.TouchIdPlugin.CheckTouchIdPlugin	0.00	0.00
NRNodes.ExecuteClientAction.ReferenceClientAction.TouchIdPlugin.TouchID	0.00	0.00
NRNodes.FeedbackMessage	5.60	2.74
NRNodes.JavascriptNode	0.98	1.15
NRNodes.TriggerEvent	0.75	0.93

C

Experimental Parameters

C.1 Experimental Model Parameters

Table C.1: Experimental Model Parameters for the node kind prediction task.

Parameter	Value
GNN type	FullGN
Input Dim [x, edge_attr, u]	[117, 18, 163]
GN Layer Output dim	90
Number of GNN Layers	6
Share Layers	Yes
Layer Normalization	Yes
Output Dim	27
Activation Function	ReLu

Table C.2: Experimental Model Parameters for the node subkind prediction task.

Parameter	Value
GNN type	FullGN
Input Dim [x, edge_attr, u]	[119, 18, 165]
GN Layer Output dim	90
Number of GNN Layers	6
Share Layers	Yes
Layer Normalization	Yes
Output Dim	196
Activation Function	ReLu

C.2 Experimental Hyperparameters

Table C.3: Experimental hyperparameters for the Node kind experiment.

Model	Parameter	Value
Centralized / Local	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Training Epochs	30
FedAvg / LG-FedAvg	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Local Training Epochs	1
	Fine-Tuning	False
FedRep	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Head Training Epochs	1
	Body Training Epochs	1
	Fine-Tuning	False
FedBABU	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Body Training Epochs	1
	Fine-Tuning	False
FedHybridAvgLG	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Local Training Epochs	1
	Fine-Tuning	False
FedHybridAvgLGDual	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	FedAvg Full Training Epochs	1
	LG-FedAvg Full Training Epochs	1
	Fine-Tuning	False

Table C.4: Experimental hyperparameters for the Node kind with fine-tuning experiment.

Model	Parameter	Value
FedAvg / LG-FedAvg	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Local Training Epochs	1
	Fine-Tuning	True
	Fine-Tuning Epochs	1
	Fine-tuning Model Part	Full
FedRep	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Head Training Epochs	1
	Body Training Epochs	1
	Fine-Tuning	True
	Fine-Tuning Epochs	1
	Fine-tuning Model Part	Full
FedBABU	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Body Training Epochs	1
	Fine-Tuning	True
	Fine-Tuning Epochs	1
	Fine-tuning Model Part	Full
FedHybridAvgLGDual	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Local Training Epochs	1
	Fine-Tuning	True
	Fine-Tuning Epochs	1
	Fine-tuning Model Part	Full

Table C.5: Experimental hyperparameters for the Node subkind experiment

Model	Parameter	Value
Centralized / **Local**	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Training Epochs	30
FedAvg / **LG-FedAvg**	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Local Training Epochs	1
	Fine-Tuning	False
FedRep	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Head Training Epochs	1
	Body Training Epochs	1
	Fine-Tuning	False
FedBABU	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Body Training Epochs	1
	Fine-Tuning	False
FedHybridAvgLGDual	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	FedAvg Full Training Epochs	1
	LG-FedAvg Full Training Epochs	1
	Fine-Tuning	False

Table C.6: Experimental hyperparameters for the new action experiment

Model	Parameter	Value
Centralized / **Local**	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Training Epochs	30
FedAvg / **LG-FedAvg**	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Local Training Epochs	1
	Fine-Tuning	False
FedRep	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Head Training Epochs	1
	Body Training Epochs	1
	Fine-Tuning	False
FedBABU	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Body Training Epochs	1
	Fine-Tuning	[False, True]
	Fine-Tuning Epochs	1
	Fine-tuning Model Part	Full
FedHybridAvgLGDual	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	FedAvg Full Training Epochs	1
	LG-FedAvg Full Training Epochs	1
	Fine-Tuning	False

Table C.7: Experimental hyperparameters for the local training epochs variation experiment

Model	Parameter	Value
FedAvg / **LG-FedAvg**	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	Local Training Epochs	[1, 5]
	Fine-Tuning	False
FedHybridAvgLGDual	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	0.001
	Selected Client Fraction	1
	Communication Rounds	30
	FedAvg Full Training Epochs	[1, 5]
	LG-FedAvg Full Training Epochs	[1, 5]
	Fine-Tuning	False

Table C.8: Experimental hyperparameters for the learning rate variation experiment

Model	Parameter	Value
FedAvg / **LG-FedAvg**	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rates	[0.1, 0.01, 0.001, 0.0001]
	Selected Client Fraction	1
	Communication Rounds	30
	Local Training Epochs	1
	Fine-Tuning	False
FedHybridAvgLGDual	Loss	Cross Entropy
	Batch Size	128
	Optimizer	SGD
	Learning Rate	[0.1, 0.01, 0.001, 0.0001]
	Selected Client Fraction	1
	Communication Rounds	30
	FedAvg Full Training Epochs	1
	LG-FedAvg Full Training Epochs	1
	Fine-Tuning	False