Federated Learning for Predicting the Next Node in Action Flows

Daniel Lopes
daniel.f.lopes@tecnico.ulisboa.pt

Instituto Superior Técnico

(Advisor: Professor Luís Rodrigues, IST)
(Co-Advisor: João Nadkarni, Outsystems)

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 access to different data, as well as personalised classification preferences, 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. We survey the state-of-the-art solutions for federated learning and identify unexplored alternatives for training the global and the local models that are worth exploring. Our research is driven by the requirements of OutSystems, where federated learning is being explored as an alternative approach to train Graph Neural Networks that are used to help programmers in their coding tasks.

Keywords: Federated Learning · Personalization · Graph Neural Networks

Table of Contents

1 Introduction . . . 3
2 Goals . . . 4
3 Background . . . 5
3.1 Machine Learning . . . 5
3.2 Federated Learning . . . 7
3.3 Communication Efficiency in Federated Learning . . . 8
3.4 Privacy of Client Data . . . 9
3.5 Security of the Model . . . 10
3.6 Personalized Federated Learning . . . 12
3.7 Federated Learning with Graph Neural Networks . . . 13
4 Related Work . . . 13
4.1 Systems Addressing Communication Efficiency . . . 13
4.2 Systems Addressing Privacy . . . 14
4.2.1 Privacy Attacks . . . 14
4.2.2 Privacy Defense Systems . . . 15
4.3 Systems Addressing Poisoning . . . 16
4.3.1 Poisoning Attacks . . . 16
4.3.2 Poisoning Defense Systems . . . 16
4.4 Systems Addressing Personalization . . . 17
5 Architecture . . . 21
5.1 OutSystems's Federated Learning Setting . . . 21
5.2 General Architecture . . . 22
5.3 Communication Efficiency . . . 23
5.4 Ensuring Privacy and Security . . . 23
6 Evaluation . . . 24
6.1 Performance . . . 24
6.2 Communication Efficiency . . . 25
6.3 Privacy and Security . . . 25
7 Scheduling of Future Work . . . 25
8 Conclusions . . . 25
A Proposed Training Procedure . . . 29
B OutSystems's Data Set Statistics . . . 30
C OutSystems's Data Set Distribution . . . 30

1 Introduction

Machine Learning (ML) is an area of Artificial Intelligence (AI) that intends to learn the parameters of a model from a given training data set such that they can be used to predict an output given an input. Federated Learning (FL) is a particular case of ML where different edge devices cooperate 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.

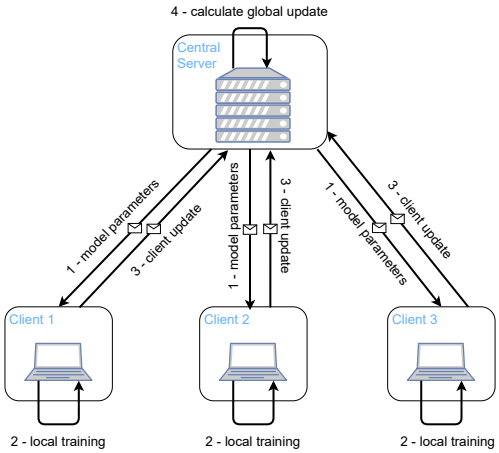

Fig. 1. Example of the steps of a single FL communication round

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 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. Aspects such as privacy preservation, security, and communication efficiency will also be considered. 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 these new variants.

We plan to 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.

In Service Studio, 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.

The rest of the report is organized as follows. Section 2 briefly summarizes the goals and expected results of our work. In Section 3, we present all the background related to our work. Section 4 covers the related work. Section 5 describes the proposed architecture to be implemented, and Section 6 describes how we plan to evaluate our results. Finally, Section 7 presents the schedule of future work, and Section 8 concludes the report.

2 Goals

The high-level goal of our work is to explore new personalization approaches for FL that, in practice, can be used by the OutSystems's Service Studio. Given that, in this application, the action flow to be modelled is represented by

a graph, we will use GNNs to perform node classification tasks. Also, since the personalization of the classification results is of paramount importance, we aim at FL algorithms that have good personalization characteristics. More precisely:

> *Goals:* We aim at exploring different techniques that divide the ML model in two parts that are combined, namely, a shared part and multiple personalized local parts, to improve the performance of FL. While doing so, we also want to study the impact of these techniques on the privacy, security, and efficiency of the resulting FL system.

To achieve this goal, we plan to leverage recent research results such as Federated Learning with Personalization Layers (*FedPer*) [1], Federated Representation Learning (*FedRep*) [9], and Federated Averaging with Body Aggregation and Body Update (*FedBABU*) [26], that already combine the use of global and local models, and that have proposed different strategies to train these models. We aim at identifying new strategies that have not been explored by these systems and assessing their potential.

The project will produce the following expected results.

> *Expected results:* The work will produce i) a specification of the system, including the proposed personalized training algorithm; ii) an implementation of the system, iii) an extensive experimental evaluation of the performance of the algorithm and, finally, iv) an analysis of its privacy-preservation and security features.

3 Background

In this section, we present some background related to our work. We start by a simple introduction to ML (Section 3.1), then we present FL as an ML approach (Section 3.2). We then go over some of the challenges of FL, namely, communication efficiency (Section 3.3), client data privacy (Section 3.4), model security (Section 3.5) and model personalization (Section 3.6). Finally, we explore some of the work done for GNNs in FL (Section 3.7).

3.1 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 allow to 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 (problem known as overfitting).

The task of building an ML model usually leverages three phases: training, validation, and testing. The data set is partitioned into three disjoint partitions, one for each stage. This division allows assessing the generalization capabilities of the model through the evaluations made in the unseen partitions. 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 patterns in the data which allow it to derive the expected output from the received input, these patterns are captured by the model. 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 (if the performance in validation data is decreasing while the performance in training data keeps increasing it is an indicator of that). 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 stop overfitting in the training stage.

There are several types of ML models. In this work, we focus on Artificial Neural Networks and, particularly, on Graph Neural Networks (GNNs) [3], as they are the architecture used by the AI-powered capabilities present in Service Studio. GNNs learn data sets that are modelled as graphs. Other neural network architectures exist 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 proposed by Battaglia et al. [3].

A graph $G = (V, E)$ is characterized by a set of nodes V and a set of edges E. In the case of OutSystems, an Action Flow, also called a Logic Flow, is a directed weakly connected graph. 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, it can be a "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 attribute indicates the condition the edge corresponds to. In an action flow, G cannot have self-loops or parallel edges, V must contain 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 have incoming edges.

Figure 2 shows an example of 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

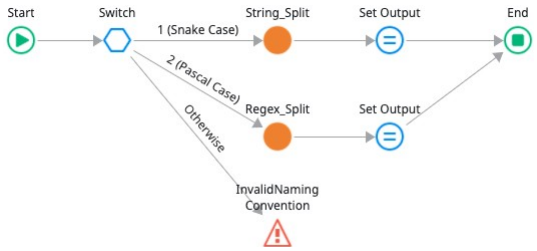

Fig. 2. Service Studio Action Flow for splitting a string into multiple tokens according to the used naming convention.

(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, 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".

3.2 Federated Learning

Traditional ML approaches assume that the entire data set is available to the workers that build the model. Even if the training is distributed, and some workers may not be able to access the entire data set, there are no restrictions that prevent specific data sets to be processed by specific workers. In contrast, in FL, 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. [6]. 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, 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*) [22] 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*) [28] 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 case we are dealing with a heterogeneous setting.

Different categories to classify FL approaches have been proposed in the literature [15,17,37]. 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 datacenters, 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.

3.3 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. [15] 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 defense strategies that we will discuss next, namely Differential Privacy, since this defense 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 computationally more efficient.

3.4 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 defenses, it is important to classify the different types of possible adversaries [5,21]:

- *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 [21,38]:

- *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 [21,33]:

– *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 3.5, 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 the local and global updates by eavesdropping the communication channels. Thus, secure communication channels must be used to mitigate this threat.

3.5 Security of the Model

In FL, several clients participate in each round and directly influence the global model. This opens the possibility for having 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 [21]:

– *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 defense mechanisms available to try and mitigate poisoning attacks can be of two types depending on the strategy used [30]:

– *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 defense 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 defenses have been shown to be vulnerable to some attacks.

It is also important to mention that Shejwalkar et al. [29] 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 defense strategy is enough to protect FL in production against untargeted attacks.

3.6 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 the clients. In order to handle the heterogeneity of client's data, FL methods were proposed with the intent of personalizing the global model to the different clients. Tan et al. [31] 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.

Data Augmentation 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.

Client Selection 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.

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

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

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

Multi-task Learning is a method where 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.

Parameter Decoupling is a technique that 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.

3.7 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 them. In terms of privacy-preservation, Jiang et al. [14] 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 4.2). Furthermore, Zhou et al. [40] propose an FL system for node classification tasks in vertical FL settings, using local DP.

In terms of personalization approaches, Wang et al. [34] propose an FL system for node classification in a semi-supervised setting, leveraging meta-learning for vertical FL. Xie et al. [36] propose an FL system for graph classification in a vertical FL setting using clustering to group similar clients through their updates.

Zhang et al. [39] 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 5.1).

4 Related Work

In this section, we present some work related to the previously mentioned FL challenges. Therefore, Section 4.1 explores communication efficiency systems, Section 4.2 covers privacy attacks and defenses, Section 4.3 covers poisoning attacks and defenses, and Section 4.4 addresses personalization systems.

4.1 Systems Addressing Communication Efficiency

Structured and Sketched updates [16] 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.

Federated Dropout [8] is a technique to reduce the costs of the 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.

Communication-Mitigated Federated Learning (CMFL) [20] 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.

4.2 Systems Addressing Privacy

4.2.1 Privacy Attacks

There are several examples in the literature that illustrate how FL can be compromised. Inference of class representatives attacks are explored by the *multi-task GAN for Auxiliary Identification (mGAN-AI)* [35] framework which leverages Generative Adversarial Networks (GANs) 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. [25], 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. [24]. 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* [41] 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.

4.2.2 Privacy Defense Systems

Secure Multiparty Computation is one of the privacy defense strategies of which the *Secure Aggregation* [7] protocol is an example of. 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 for 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 defenses (Section 3.5).

Homomorphic Encryption is another possible defense strategy. Phong et al. [27] 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.

Differential Privacy is the last defense strategy. Geyer et al. [13] 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. [23], 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 to the server, poisoning defense techniques can be used.

4.3 Systems Addressing Poisoning

4.3.1 Poisoning Attacks

Several poising attacks have been identified in the literature. Data Poisoning attacks are illustrated by Tolpegin et al. [32] 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. [2] 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.

4.3.2 Poisoning Defense Systems

Krum [4] 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.

Foolsgold [12] 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 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.

4.4 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).

Personalized Federated Averaging (Per-FedAvg) [11] 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 [9]); and is more costly in terms of communication, since the full model is exchanged.

FedProx [18] 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.

FedU [10] 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.

Federated Learning with Personalization Layers (FedPer) [1] can be classified as a multi-model joint-training head-custom (MJH) algorithm based on parameter decoupling. 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, leading to worse performance. 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.

Local Global Federated Averaging (LG-FedAvg) [19] can be classified as a multi-model disjoint-training body-custom (MDB) algorithm that takes the opposite approach from *FedPer*. 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, each client's local computation round consists of two *SGD* steps: on the first, the local body is trained; on the second, the global head is trained, using the newly trained body. After performing all the local computation rounds, the client sends the updated head 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. [9], indicating that personalizing the head may be better than personalizing the body.

Federated Representation Learning (FedRep) [9] can be classified as a multi-model disjoint-training head-custom (MDH) approach similar to *FedPer* but with a slight change. 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 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*. 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.

Federated Averaging with Body Aggregation and Body Update (FedBABU) [26] can be classified as a multi-model disjoint-training head-custom (MDH) algorithm based on parameter decoupling. 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 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 by 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.

Table 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. Also, we present two alternatives that, to the best of our knowledge, have not yet been explored in the literature. In our work, we plan to assess the advantages

Table 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	MDB	Parameter Decoupling	N	head	body	$2E$	body first (w/ global head) head last (w/ trained body)	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 only)
Alternative 1	MDH	Parameter Decoupling	N	body	head	$E_H + E_B$	head first (w/ global body) body last (w/ fixed head)	optional after training (full model)
Alternative 2	MDH	Parameter Decoupling	N	body	head	$E_B + E_H$	body first (w/ fixed head) head last (w/ trained body)	optional after training (full model)

and limitations of these alternatives, as we will describe in detail in the next section.

5 Architecture

In this section, we elaborate on the proposed system. Therefore, we start by defining the FL setting assumed, then, we present the proposed FL system, including the suggested personalization algorithm. The presented system represents the core of our work, however, as a secondary objective, and only if enough time is available, we also propose some communication efficiency improvement techniques and some privacy and security defenses to be implemented.

5.1 OutSystems's Federated Learning Setting

Before detailing our proposed FL solution, we first need to properly characterize the environment for which such solution will be developed. In our setting, the clients are the machines of the developers using the OutSystems's Service Studio. The trained model should be able to give predictions for possible next nodes of an action flow, and for that, a GNN model is used. In particular, the model's objective is to predict one of the nodes' attributes: the node kind.

Since the OutSystems platform has a limited number of possibilities for the types of nodes in an action flow, the possible feature values for the "kind" attribute of the nodes are also the same. However, each client has its own way of coding, meaning action flows will be different for each client, for instance, one client might use a "switch" node whereas another might use a chain of "if" nodes to test for the correct condition. This difference between clients' action flows means we are dealing with a horizontal FL setting. Furthermore, we assume the data for each client not to be representative of the data of other clients, since each one has its own unique action flows. This means that our setting is highly heterogeneous, therefore, we are dealing with non-i.i.d. data. Also, the data is unbalanced since each client can have a significantly different number of data points (in our case, action flows) in his 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 machines of developers, thus, we expect a large number of clients to participate, meaning we are working in a cross-device setting. Having enterprise machines as end devices 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.

5.2 General Architecture

In Section 4.4, we explored some of the personalized FL algorithms that have been proposed in the literature. We base our approach on two of them: *FedRep* and *FedBABU*. Both these algorithms have been recently proposed and they seem to report the most promising performance out of all the seen algorithms.

As seen before, *FedRep* argues that heterogeneity resides in the classification and a common representation of the clients' data can be obtained through client collaboration. Therefore, it trains the head and the body of the model separately, each possibly with a different number of local steps. On the other hand, *FedBABU* argues that the training of the global representation is affected negatively by the training of the local classifier. Thus, it proposes fixing the classifier during the training process in order to achieve the best representation possible.

Our approach can be seen as a hybrid of the two algorithms, where we try to obtain the best representation possible by training it with a fixed classifier as in *FedBABU* while also allowing clients to train a personalized classifier using the received global representation. This client-specific classifier is kept locally and not used in the training of the local representation. By doing this, we follow the approach of *FedRep*, where the head is trained separately from the body and each one can have a different number of local training rounds.

The protocol proceeds in communication rounds similarly to *FedAvg*. The server procedure is identical to *FedBABU*'s, thus, the server first initializes the global model. In each communication round, the server selects a random subset of clients to participate and sends them the model parameters of the previous

round, composed by the body of the previous round, and the fixed head to be used for local training of the body. The server then waits for the local updates of the clients, that is, the client bodies after training with the local data. After receiving the updates, the server performs a weighted average considering the number of data points in each participating client's data set to obtain the aggregated body to be used in the following communication round.

In terms of the training procedure of each client, we propose two alternatives: (1) the head is trained first; (2) the body is trained first. Appendix A contains the pseudo-code for the proposed training procedure. In both of the alternatives, the client maintains its personalized head. For (1), the personalized head of the client is trained first by performing E_H local training rounds executing SGD with the received global body. Then, the body received from the server is trained with the fixed head for E_B local rounds, after which the trained body is returned to the server. Alternative (2) follows the opposite procedure, it first trains the global body for E_B rounds with the fixed head and then trains the local head with the newly trained local body, returning the trained body to the server after finishing. In summary, the difference between the two alternatives is the local head being trained with the global body, (1), or with the locally trained body, (2), thus for the latter the obtained head is more specialized since the body used was previously trained. Table 1 summarizes both alternatives (1) and (2) (rows "Alternative 1" and "Alternative 2", respectively).

Through this training procedure, we can have the personalized models readily available for inference after the training phase, without needing to perform unnecessary computation rounds for fine-tuning. Furthermore, the classifier is constantly adapted to the global representation, instead of being personalized from a fixed classifier as in *FedBABU*, meaning it can better fit the representation. Finally, it allows for training the representation without the influence of the classifier which could degrade the quality of the obtained global representation.

5.3 Communication Efficiency

The proposed approach already reduces the communication costs when compared to *FedAvg* since only the body of the model is exchanged. Furthermore, in our setting (Section 5.1) the communication costs are not as relevant. Nonetheless, if we have enough time, we might experiment with some cost reduction methods to reduce the number of parameters sent/received by the clients. Some of the possible approaches are covered in Section 3.3.

5.4 Ensuring Privacy and Security

Before defining the defense 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 the local updates and the global updates if the communication channels are not secure.

In terms of the security and privacy of the proposed training algorithm, we argue that it does not introduce new security or privacy threats in comparison to *FedAvg*, since the attack surface is the same for both approaches. The only differences between the two reside in how the local computation on the client is performed and on what part of the model is aggregated, both of which do not increase the attacker power. Interestingly, since we only exchange the body, the attackers cannot manipulate the head, limiting the attacker power.

As an additional objective, we propose to apply some defense mechanisms. These 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. Therefore, TLS should be used for secure communication channels to prevent eavesdroppers. Server-side DP should also be used to add noise to the global update such that clients do not have access to the fully denoised aggregated update. Finally, an anomaly detection mechanism should be used to try to prevent poisoning attacks, for instance, *Foolsgold*, which can also be enhanced by a robust aggregation algorithm.

6 Evaluation

The system will be evaluated primarily regarding the performance obtained using the proposed training procedure. However, if enough time is available for the implementation of the secondary objectives, the communication efficiency and/or privacy and security guarantees of the system will also be evaluated.

The experiments will be run in the AWS cloud with a proprietary OutSystems data set. This data set is composed of 986 real-world codebases from the OutSystems platform, where each codebase contains all the applications of one client. Each application is modelled as a graph that can contain multiple Action Flows among other useful information. Our focus is on the action flow subgraphs of each application graph as the objective of the trained model is to predict one of the attributes of the next node of these subgraphs. Some statistics about the data set are provided in Appendix B as well as the distribution graph of the number of action flows per client in Appendix C. From these, we can clearly see how widely spread in terms of action flows per client the dataset is as the standard deviation is rather high. Moreover, each code base will be divided into three partitions, one for each stage of the ML model building process (training, validation, testing).

6.1 Performance

The performance will be evaluated by comparing the accuracy obtained through our proposed training procedure with some of the existing personalization algorithms as well as, and most importantly, with the current centralized OutSystems's algorithm and with locally trained client models. From previous

results, we know that the locally trained models obtain higher accuracy than the centralized model. Therefore, ideally, our goal is for our approach to obtain higher accuracy than the local models. Nonetheless, we also consider having achieved encouraging results if the accuracy obtained stands in-between these two approaches since we can achieve better performance than the current centralized model while giving clients better privacy guarantees through FL.

Furthermore, it is also interesting to test the accuracy for new clients, since the main disadvantage of locally trained models is clients not having enough data to extract data relationships, favouring our approach in this particular case since those relationships would be supplied by the global model. Finally, we also intend to test how the performance is affected by varying the different parameters, for instance, the number of clients, the number of local computations for the head/body, and the data heterogeneity.

6.2 Communication Efficiency

The communication efficiency will be evaluated to understand the communication costs of our approach. We intend to evaluate whether communication reduction techniques can reduce the communication overhead while still maintaining a good model performance. The efficiency of the communication will be measured in terms of the number of model parameters sent through the network.

6.3 Privacy and Security

We also intend to measure the privacy and security of the proposed approach with and without defense mechanisms. Therefore, we intend to perform some of the attacks existent to assess how much information the attacker can obtain from the global updates and how much can the attacker alter the global model.

7 Scheduling of Future Work

Future work is scheduled as follows:

- January 15 - March 29: Detailed design and implementation of the proposed architecture, including preliminary tests.
- March 30 - May 3: Perform the complete experimental evaluation of the results.
- May 4 - May 23: Write a paper describing the project.
- May 24 - June 15: Finish writing the dissertation.
- June 15 Deliver the MSc Dissertation.

8 Conclusions

FL is an ML approach that allows clients to collaboratively train a global model with their own private data without its privacy being compromised. However, there are some challenges that emerge when developing an FL system.

In this report, we surveyed some solutions proposed for the challenges of FL and based on the analyzed solutions, we propose a personalization training algorithm to handle data heterogeneity in an attempt to train models that achieve better performance than both the current OutSystems centralized algorithm and locally trained models. We also explore some communication efficiency improvement techniques and some defense mechanisms, to ensure the privacy and security of an FL system. Finally, we detail the methodology to be used to evaluate the FL system to be implemented and finish by scheduling the future work.

Acknowledgments We are grateful to Filipe Assunção, Miguel Lopes and Afonso Gonçalves for the fruitful discussions and comments during the preparation of this report. This work was done in the scope of a curricular internship at OutSystems.

References

1. Arivazhagan, M.G., Aggarwal, V., Singh, A.K., Choudhary, S.: Federated learning with personalization layers. CoRR **abs/1912.00818** (December 2019)
2. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 108, pp. 2938–2948. PMLR, Virtual Event (August 2020)
3. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al.: Relational inductive biases, deep learning, and graph networks. CoRR **abs/1806.01261** (October 2018)
4. Blanchard, P., El Mhamdi, E.M., Guerraoui, R., Stainer, J.: 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. pp. 119–129. Curran Associates, Inc., Long Beach, CA, USA (December 2017)
5. Blanco-Justicia, A., Domingo-Ferrer, J., Martínez, S., Sánchez, D., Flanagan, A., Tan, K.E.: Achieving security and privacy in federated learning systems: Survey, research challenges and future directions. Engineering Applications of Artificial Intelligence **106**, 104468 (November 2021)
6. Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, H.B., et al.: Towards federated learning at scale: System design. In: Proceedings of Machine Learning and Systems 2019, MLSys 2019. pp. 374–388. mlsys.org, Stanford, CA, USA (March 2019)
7. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1175–1191. ACM, Dallas, TX, USA (October 2017)
8. Caldas, S., Konečny, J., McMahan, H.B., Talwalkar, A.: Expanding the reach of federated learning by reducing client resource requirements. CoRR **abs/1812.07210** (January 2019)

9. Collins, L., Hassani, H., Mokhtari, A., Shakkottai, S.: Exploiting shared representations for personalized federated learning. In: Proceedings of the 38th International Conference on Machine Learning, ICML 2021. Proceedings of Machine Learning Research, vol. 139, pp. 2089–2099. PMLR, Virtual Event (July 2021)
10. Dinh, C.T., Vu, T.T., Tran, N.H., Dao, M.N., Zhang, H.: A new look and convergence rate of federated multi-task learning with laplacian regularization. CoRR **abs/2102.07148** (December 2021)
11. Fallah, A., Mokhtari, A., Ozdaglar, A.: 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. pp. 3557–3568. Curran Associates, Inc., Virtual Event (December 2020)
12. Fung, C., Yoon, C.J., Beschastnikh, I.: Mitigating sybils in federated learning poisoning. CoRR **abs/1808.04866** (July 2020)
13. Geyer, R.C., Klein, T., Nabi, M.: Differentially private federated learning: A client level perspective. CoRR **abs/1712.07557** (March 2018)
14. Jiang, M., Jung, T., Karl, R., Zhao, T.: Federated dynamic gnn with secure aggregation. CoRR **abs/2009.07351** (September 2020)
15. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al.: Advances and open problems in federated learning. CoRR **abs/1912.04977** (March 2021)
16. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. CoRR **abs/1610.05492** (October 2017)
17. Li, Q., Wen, Z., Wu, Z., Hu, S., Wang, N., Li, Y., Liu, X., He, B.: A survey on federated learning systems: vision, hype and reality for data privacy and protection. IEEE Transactions on Knowledge and Data Engineering (November 2021)
18. Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V.: Federated optimization in heterogeneous networks. In: Proceedings of Machine Learning and Systems 2020, MLSys 2020. vol. 2, pp. 429–450. mlsys.org, Austin, TX, USA (March 2020)
19. Liang, P.P., Liu, T., Ziyin, L., Allen, N.B., Auerbach, R.P., Brent, D., Salakhutdinov, R., Morency, L.P.: Think locally, act globally: Federated learning with local and global representations. CoRR **abs/2001.01523** (July 2020)
20. Luping, W., Wei, W., Bo, L.: Cmfl: Mitigating communication overhead for federated learning. In: 39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019. pp. 954–964. IEEE, Dallas, TX, USA (July 2019)
21. Lyu, L., Yu, H., Ma, X., Sun, L., Zhao, J., Yang, Q., Yu, P.S.: Privacy and robustness in federated learning: Attacks and defenses. CoRR **abs/2012.06337** (December 2020)
22. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR, Fort Lauderdale, FL, USA (April 2017)
23. McMahan, H.B., Ramage, D., Talwar, K., Zhang, L.: Learning differentially private recurrent language models. In: 6th International Conference on Learning Representations, ICLR 2018. OpenReview.net, Vancouver, BC, Canada (April 2018)
24. Melis, L., Song, C., De Cristofaro, E., Shmatikov, V.: Exploiting unintended feature leakage in collaborative learning. In: 2019 IEEE Symposium on Security and Privacy, SP 2019. pp. 691–706. IEEE, San Francisco, CA, USA (May 2019)

25. Nasr, M., Shokri, R., Houmansadr, A.: 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. pp. 739–753. IEEE, San Francisco, CA, USA (May 2019)
26. Oh, J., Kim, S., Yun, S.Y.: Fedbabu: Towards enhanced representation for federated image classification. CoRR **abs/2106.06042** (June 2021)
27. Phong, L.T., Aono, Y., Hayashi, T., Wang, L., Moriai, S.: Privacy-preserving deep learning via additively homomorphic encryption. IEEE Transactions on Information Forensics and Security **13**(5), 1333–1345 (May 2018)
28. Ruder, S.: An overview of gradient descent optimization algorithms. CoRR **abs/1609.04747** (June 2017)
29. Shejwalkar, V., Houmansadr, A., Kairouz, P., Ramage, D.: Back to the drawing board: A critical evaluation of poisoning attacks on federated learning. CoRR **abs/2108.10241** (December 2021)
30. Sun, Y., Ochiai, H., Esaki, H.: Decentralized deep learning for mobile edge computing: A survey on communication efficiency and trustworthiness. IEEE Transactions on Artificial Intelligence **1** (December 2021)
31. Tan, A.Z., Yu, H., Cui, L., Yang, Q.: Towards personalized federated learning. CoRR **abs/2103.00710** (March 2021)
32. Tolpegin, V., Truex, S., Gursoy, M.E., Liu, L.: Data poisoning attacks against federated learning systems. In: Computer Security – ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020. Lecture Notes in Computer Science, vol. 12308, pp. 480–501. Springer, Guildford, UK (September 2020)
33. Truong, N., Sun, K., Wang, S., Guitton, F., Guo, Y.: Privacy preservation in federated learning: An insightful survey from the gdpr perspective. Computers and Security **110**, 102402 (November 2021)
34. Wang, B., Li, A., Li, H., Chen, Y.: Graphfl: A federated learning framework for semi-supervised node classification on graphs. CoRR **abs/2012.04187** (December 2020)
35. Wang, Z., Song, M., Zhang, Z., Song, Y., Wang, Q., Qi, H.: Beyond inferring class representatives: User-level privacy leakage from federated learning. In: 2019 IEEE Conference on Computer Communications, INFOCOM 2019. pp. 2512–2520. IEEE, Paris, France (April 2019)
36. Xie, H., Ma, J., Xiong, L., Yang, C.: Federated graph classification over non-iid graphs. CoRR **abs/2106.13423** (November 2021)
37. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology (TIST) **10**(2), 12:1–12:19 (January 2019)
38. Yin, X., Zhu, Y., Hu, J.: A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions. ACM Computing Surveys (CSUR) **54**(6), 131:1–131:36 (July 2021)
39. Zhang, K., Yang, C., Li, X., Sun, L., Yiu, S.M.: Subgraph federated learning with missing neighbor generation. CoRR **abs/2106.13430** (November 2021)
40. Zhou, J., Chen, C., Zheng, L., Wu, H., Wu, J., Zheng, X., Wu, B., Liu, Z., Wang, L.: Vertically federated graph neural network for privacy-preserving node classification. CoRR **abs/2005.11903** (April 2021)
41. Zhu, L., Liu, Z., Han, S.: Deep leakage from gradients. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019. pp. 14747–14756. Curran Associates, Inc., Vancouver, BC, Canada (December 2019)

A Proposed Training Procedure

Algorithm 1 Proposed Training Procedure.

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**
    $w_{k,body}^t \leftarrow \text{ClientLocalUpdate}(w_G^{t-1})$
  $n \leftarrow \sum_{i=1}^m n_i^t$
  $w_{G,body}^t \leftarrow \sum_{i=1}^m \frac{n_i^t}{n} w_{i,body}^t$

  $w_G^t \leftarrow \{w_{G,body}^t, w_{G,head}^0\}$
procedure ClientLocalUpdate(w_G^{t-1}) ▷ Client executes
  $\{w_{G,body}^{t-1}, w_{G,head}^0\} \leftarrow w_G^{t-1}$
  $w_{k,body} \leftarrow w_{G,body}^{t-1}$ ▷ $w_{k,head}$ is maintained locally
  if head trained first **then** ▷ 1st alternative
    for each $1, 2, \dots, E_H$ **do**
      $w_{k,head} \leftarrow SGD(w_{k,body}, w_{k,head}, \mathcal{D}_k, B)$
    for each $1, 2, \dots, E_B$ **do**
      $w_{k,body} \leftarrow SGD(w_{k,body}, w_{G,head}^0, \mathcal{D}_k, B)$
  else if body trained first **then** ▷ 2nd alternative
    for each $1, 2, \dots, E_B$ **do**
      $w_{k,body} \leftarrow SGD(w_{k,body}, w_{G,head}^0, \mathcal{D}_k, B)$
    for each $1, 2, \dots, E_H$ **do**
      $w_{k,head} \leftarrow SGD(w_{k,body}, w_{k,head}, \mathcal{D}_k, B)$
  returns $w_{k,body}$

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 $w_{G,head}^0$ the initial head parameters (the fixed head);
- $w_{i,body}^t$ represents the local body 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_B represents the number of local computation epochs for the body and E_H for the head;
- $w_{k,body}$ 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 obtained from the data set of client k, $\mathcal{D}_k$, for a model with body parameters w_b and head parameters w_h.

B OutSystems's Data Set Statistics

Table 2. Statistics on the number of action flows per client in the OutSystems data set.

min	8
max	54252
mean	4244.879
median	2289.500
var	39079685.969
std	6251.375
25 percentile	918.000
75 percentile	5020.250
90 percentile	9894.500
95 percentile	14561.000
99 percentile	32535.450

C OutSystems's Data Set Distribution

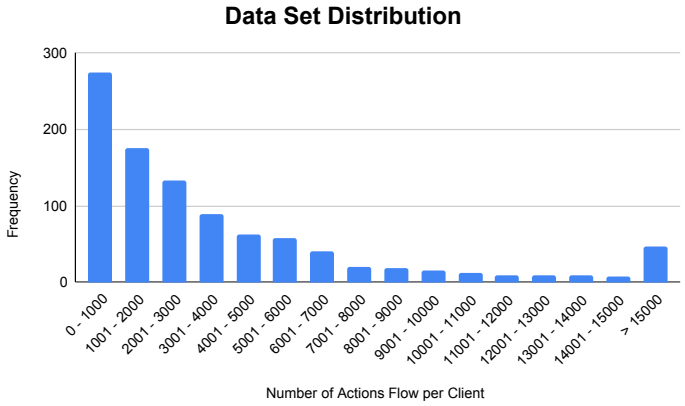

Fig. 3. OutSystems's number of Action Flows per client frequency distribution with bin size of 1000