Introducing Federated Learning into Internet of Things ecosystems – preliminary considerations

Karolina Bogacka  
Systems Research Institute  
Polish Academy of Sciences  
Warsaw, Poland  
0000-0002-7109-891X

Katarzyna Wasielewska-Michniewska  
Systems Research Institute  
Polish Academy of Sciences  
Warsaw, Poland  
0000-0002-3763-2373

Marcin Paprzycki  
Systems Research Institute  
Polish Academy of Sciences  
Warsaw, Poland  
0000-0002-8069-2152

Maria Ganzha, Anastasiya Danilenka  
Warsaw University of Technology  
Warsaw, Poland  
0000-0001-7714-4844, 0000-0002-3080-0303

Lambis Tassakos  
TwoTronic Gmbh  
Meitingen, Germany  
0000-0003-2511-9035

Eduardo Garro  
Prodevelop  
Valencia, Spain  
0000-0002-8160-0125

Abstract—Federated learning (FL) was proposed to train models in distributed environments. It facilitates data privacy and uses local resources for model training. Until now, the majority of research has been devoted to the “core issues”, such as adaptation of machine learning algorithms to FL, data privacy protection, or dealing with effects of unbalanced data distribution. This contribution is anchored in a practical use case, where FL is to be actually deployed within an Internet of Things ecosystem. Hence, different issues that need to be considered are identified. Moreover, an architecture that enables the building of flexible, and adaptable, FL solutions is introduced.

Index Terms—applied federated learning, Internet of Things, federated learning topology

I. INTRODUCTION

One of the critical (and practical) bottlenecks of the application of Machine Learning (ML) lies in the limited ability to collect, consistently label, and use large datasets. This is particularly the case for businesses that do not possess “unlimited resources”, such as Google or Amazon do [1]. Moreover, while existing data may be large and labeled, it may also be “split between stakeholders”, who do not want and/or cannot share their datasets [2], e.g. as in the case of medical data, which belongs to different hospitals/clinics. Moreover, ongoing controversies concern the collection and storage of information [3]. However, many ML developments, e.g. in mobile applications, rely on models being periodically (re/up)trained on sensitive private data (e.g., browsing history, or geo-positioning). Hosting such data in a centralized location, even in adherence to strict legislation, still poses serious security risks, as can be seen through repeated data leaks [4]–[6]. Note also that the latest advancements in ML involve training very large models and thus require enormous computational resources [7]. This not only increases the cost but also the carbon footprint [8].

To overcome these, and other related, problems, Federated Learning (FL) has been proposed [9]. Its name comes from the use of a flexible federation of collaborating (often heterogeneous, edge) devices, known as clients, “synchronized” and “orchestrated” by a “central server”. In FL, (i) clients train copies of the global model, using local data, and (ii) update its local model back to the server to continue the process, until a stopping criterion is met. Therefore, private data never leaves the clients [11]. While a lot of research is devoted to the FL process itself, it is mostly implemented and tested in the cloud. Hence, important practical issues are omitted [12], [13].

One should realize that the Internet of Things (IoT) is one of the future areas of FL application. This follows the general trend to replace cloud-centric solutions with edge-cloud continuum approaches [14]. This, in turn, is happening because storing data, and providing resources in the data center, is not sustainable for large-scale complex deployments, where latency can negatively impact performance. Hence, computing has to take place near (at) the edge, i.e. physically close to sensors and/or users [15]. The resulting edge-cloud continuum ecosystem is seen as the necessary direction for the evolution of the Next-Generation Internet of Things [13]. Here, among others, FL will deliver intelligence at the edge [10]. However, combining FL with IoT brings about practical issues: (i) heterogeneity of clients and networks can cause delays (latency variability), or introduce “stragglers” (weaker/more busy clients), slowing down the training process; (ii) computing and/or storing resources on the (far) edge devices, and their battery life, tend to be very limited, impeding the use of large models and imposing restrictions on training time; and (iii) data used for the training can be highly redundant [12].

As noted, core FL research is focused on machine learning (ML) and its intricacies. This can also be seen in the state-of-

Work supported by ASSIST-IoT project funded from the European Union’s Horizon 2020 RIA program under grant 957258. Work of Maria Ganzha and Anastasiya Danilenka is funded in part by the Centre for Priority Research Areas Artificial Intelligence and Robotics of Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) programme.
the-art of FL frameworks. For example, though TensorFlow Federated Framework (TFF) [16] offers a variety of stable ML models, it supports experimentation only in a simulated environment. In other words, TFF currently does not work with actual “edge devices”. Similarly, the FATE [17] platform requires 6GB of RAM, and 100 GB of disk space, on the server and on the clients. While this would work in a laboratory, it exceeds the capabilities of the majority of edge devices (of today). Among platforms, PaddleFL enables the implementation of decentralized architectures by default. However, since PaddleFL is not a very popular platform ([18]), it may not be an optimal choice for production deployments. Flower platform ([19]) can run on a diverse range of environments and devices, including Android, iOS, Raspberry Pi, and Nvidia Jetson. It is also compatible with popular ML frameworks like PyTorch and Keras. Similarly, since recently, PySyft [20] supports the use of clients on the edge with pygrid. However, even the last two platforms are, primarily, tools for studying the “nature of FL”, rather than deploying FL in IoT ecosystems.

In this context, this contribution (a) reflects the nature of challenges that actual FL deployments in IoT have to address, (b) shows how a reference architecture, proposed for Next-Generation IoT, supports FL deployment, and (c) illustrates the flexibility of the proposed approach through its capability of setting systems with different FL topologies.

Hence, the remaining parts of this work are organized as follows. In Section II a practical IoT-based scenario from ASSIST-IoT 1 project is described. Since FL will be actually deployed and experimented with in this use case, it will be used to summarize key requirements for “practical FL in IoT”. In Section III pertinent state-of-the-art is summarized. Following, in Section IV, an architecture that fulfills the requirements of the use case and addresses issues materializing in IoT-based deployments is described. Next, in Section V, the ways of adapting and using the proposed architecture are outlined. Finally, in Section VI, a summary of contributions, and directions for future work are provided.

II. FEDERATED LEARNING USE CASE IN IOT DEPLOYMENT

The core of this work is provided by the ASSIST-IoT project. There, a sample use case, in which FL is to be applied, is a part of the car damage recognition pilot scenario. Here, the goal is to provide fast and accurate inspection of car exterior damage, with minimal data transfer from edge devices to the cloud. The task of image processing in car damage detection can be separated into: (i) separating the vehicle from the background, (ii) vehicle part segmentation, and (iii) automatic defect detection. The results are then used to support expert-delivered evaluation and facilitate decisions involving insurance claims, rental car returns, or leasing services.

As it can be seen in Fig. 1, the functional pipeline involves multiple professional scanners, equipped with high-quality cameras, based on the TwoTronic solution 2. A high volume (more than 200) of scanned vehicles per day is expected. TwoTronic scanners, and “attached” medium-class computers, will serve as FL clients. The FL server will be located in an external data center in Nürnberg, Germany.

![Car damage recognition – scanner gate](image)

Deploying an FL system is a complex task. Importantly, to practically apply FL solution in this real-life use case, additional issues, rarely addressed in FL literature, have to be addressed: (a) sudden user dropout, (b) weak network connection with potential interruptions, (c) geographical constraints (leading to unequal groups of clients), (d) data distribution (local client distribution differing from global distribution, with no additional public information to mitigate the problem through client grouping), (e) system limitations, i.e. locally available RAM and number of cores, and (f) the fact that heterogeneous devices may bring about interoperability issues.

Note that, due to (geographical) distances between scanners and the FL server (located in Nürnberg), and the need for high speed and accuracy of prediction, examining different FL topologies may be in order. First, the introduction of additional clients fulfilling the functionality of local servers could mean, that more information about the interrupted training processes is preserved. Second, training could continue within lower levels of aggregating hierarchy. Additionally, a decrease in

---

1 https://assist-iot.eu/
2 https://www.fahrzeugscanner.de/
direct communication between scanners and the central server could lead to faster training and a lesser need for expensive, high network bandwidths, reducing cost of the process. Finally, application of non-standard topologies may reduce training interruptions and client dropouts, by introducing additional communication channels.

III. WORK RELATED TO FEDERATED LEARNING TOPOLOGY

Taking into the account potential importance of FL topology, let us summarize related state-of-the-art. Currently, the effect of topology between clients on FL systems is not fully understood, but hard to deny [21]. Overall, there is no “best topology”, but it needs to be selected to match the needs of a specific use case. It has been observed that the centralized approach may not always be appropriate, due to communication overhead and a single point of failure [22]. On the other hand, fully decentralized topologies involve a significant cost of communication, not related to client-server connections [23]. Finally, some works combine these approaches to improve convergence and scalability, e.g. by combining decentralized groups with a centralized update schema [24].

Some approaches experimented with combinations of star and ring architectures, to avoid the communication bottleneck of the former while gaining improved scalability and accuracy of the latter. In [25], a star architecture with ring-based groups, supported by a self-balancing framework designed to mitigate the problem of a skewed global distribution, was evaluated. Work presented in [26] uses a ring architecture with star-based groups for non-IID data. Overall, while a linear speedup with respect to the number of clients is reported, the need for periodic variance in data is a limiting factor.

Ring-based groups, without global communication, trained using semi-cyclic Stochastic Gradient Descent are discussed in [27]. Here, it is observed that the use of ring-based groups may lead to slower training, due to the higher number of rounds the process has to undergo for the model to gather information from all the nodes when compared with star-based groups. Note, that not all applications are training time critical so this feature should not be treated as a definite drawback.

Work reported in [24] combines star and ring architectures and proposes two forms of the TornadoAggregate algorithm: (1) a ring architecture with star-based groups, and (2) a star architecture with ring-based groups. Interestingly, a substantial star architecture and ring-based groups outperformed the ring architecture with star-based groups, in terms of the final model accuracy while preserving the same scalability. Other performance indicators, however, have not been provided.

In [21] D-Cliques, a topology that aims to reduce gradient bias by grouping clients in sparsely interconnected cliques, such that the label distribution in the clique would be representative of the global distribution, is presented. This approach led to a convergence speed similar to that of a fully-connected topology with a 98% reduction in the total number of edges, and a 96% reduction in the total number of messages.

A contrasting approach to data skewness mitigation is presented in [28], in the form of a hierarchical FL system with Federated Gradient Descent conducted on the user-edge layer, and Federated Averaging on the edge-cloud layer. The resulting architecture is designed for an IoT environment, acknowledging that the potentially less efficient connections between the user and the server will most likely support less frequent communication than the edge-server links.

Work described in [29] uses model segmentation for large model training on the far edge. The proposed approach relies on a combination of model segmentation level synchronization mechanisms, which divide the model into non-overlapping subsets, and a decentralized design, reminiscent of the gossip protocol, with each worker randomly transferring the model segment to a few other workers. Model redundancy was needed to ensure convergence, while the problem of workers suddenly exiting and returning was acknowledged. This work has been further extended in [23], forming a bandwidth-aware solution by greedily choosing clients with sufficient bandwidth to avoid delays. The convergence guarantees were provided, with the training time being reduced up to 18 times, compared to the baseline, with no accuracy degradation.

Another approach to decentralized FL (DAFCL) can be found in [30]. In DAFCL, all clients are connected through an undirected graph. Each of them trains the model based on its local data and exchanges the results with its neighbors through a symmetric doubly stochastic matrix. To avoid a single point of failure, the average model estimation is tracked using First Order Dynamic Average Consensus (FODAC). This architecture shows promising results. Nevertheless, further work on communication efficiency and increasing resilience to sudden catastrophic events (such as user dropout) would be necessary to use it in the IoT ecosystems.

In summary, research related to FL topology introduces a multitude of approaches to the problem. From the perspective of this contribution, it “does not matter” which topology should be used or is the best in a given scenario. The question is: how to make sure that any needed topology can be instantiated in the Next Generation IoT Ecosystems? Proposing a pathway to answering this question is the goal of the remaining parts of this work.

IV. FEDERATED LEARNING IN IoT – PROPOSED ARCHITECTURE

Let us now introduce the proposed architectural approach to Federated Learning in IoT ecosystems. Since support for different topologies has been shown to be important in large-scale real-life deployments, the possibility of easily implementing them is crucial. Moreover, the proposed architecture should be resistant to sudden user dropout, network connection with interruptions, or uneven grouping of clients.

The proposed FL architecture is developed according to the Reference Architecture (RA) introduced in the ASSIST-IoT project, and motivated by real-life scenarios originating from three industrial pilots. This RA is based on the concept of encapsulation, which is instantiated in the form of enablers.
FL Orchestrator is formed by four enablers: FL Orchestrator, FL Repository, FL Training Collector, and FL Local Operations. The FL Orchestrator is responsible for the configuration propagation to other enablers, workflow management, and control over the FL life cycle. It also acts as the entrance gate for human interactions. Moreover, FL Orchestrator may control the FL training process and constraints related to, e.g., the minimum number of clients or the minimum system requirements. Next, the FL Repository is a supplementary enabler for storing models, algorithms and any data needed in the FL process. Finally, the FL Training Collectors and FL Local Operations act as FL servers and clients, respectively. They are used in the constructed system as communicating components, remaining in constant contact, according to the gRPC protocol, by utilizing functionalities implemented as a part of the Flower library [33]. In other words, the FL Training Collector possesses the capabilities of an FL centralized server, while the FL Local Operations (located on edge clients) has the abilities of an FL client, with the focus placed on local model training and dataset loading. Let us now describe the FL Training Collector and the FL Local Operations in more detail.

A. FL Training Collector

FL Training Collector plays the role of the server node. It initiates the training process by uploading a configuration (e.g. from the FL Repository). The configuration data can include, among others, the specific aggregation algorithm used for FL, the minimal number of clients necessary to start training, the minimal number of clients necessary to continue training in the next round, the fraction of clients to be sampled for training or evaluation, the value of timeout for the responses from clients, the number of clients to choose for the training with blacklisting and some additional values used for later testing. The behaviour exhibited by the FL Training Collector before and after each training round, and each evaluation round, is defined as a Strategy class, matching the requirements of the Flower library. This class is used, by the Flower server, to: (i) select clients and specify weights and parameters they should receive, (ii) define mechanisms used to aggregate results, and (iii) evaluate model performance. Due to its periodic nature (methods called in a defined order, before and after every round), this class is also used to gather metrics and save current model weights. The metrics, gathered after each training round, consist of aggregated evaluation loss, global evaluation loss, and global accuracy, and are collected to monitor the training process. Later, they are locally stored in the enabler, as a serialized object inside a pickle file [34].

B. FL Local Operations

An instance of the FL Local Operations, an analogue for the FL client, is created similarly to the FL Training Collector. To start the training, it needs to be provided with a training configuration, and the address of the FL Training Collector instance, with which it should form a connection.

The FL Local Operations enabler is responsible for loading and pre-processing the right subset of local data, and setting up the local model. It not only executes but also enhances the behaviour of an FL client in the form of classes extending the flower.client.Client class, by implementing methods of initializing, fitting the model, and evaluating the resulting performance. The evaluation accuracy (which indicates how accurate are the model predictions after any given round of training) and loss (which evaluates how well the algorithm models the featured data set) of the current model are automatically computed on the local test set. The values of these metrics, in their original form, as well as an average (in the case of clustered architecture – weighted average, in an attempt to increase the precision of the visualization for unstable client groupings) over the metric values from all FL Local Operations, are used to assess the efficiency of the training process. Similarly to FL Training Collector, these statistics are regularly stored as pickle files [34]. FL Local Operations may also include mechanisms related to privacy, such as data encryption or differential privacy [4], [5].

C. FL training process

Let us now describe the FL training process that is to take place in the case of a basic centralized topology.

1) An instance of the FL Training Collector receives a training configuration from the FL Orchestrator.
2) The FL Training Collector waits for a minimal number of clients, as specified by the configuration.
3) Required number of the FL Local Operations instances receive their training configuration, from the FL Orchestrator, similar in content to that supplied to the FL Training Collector, but also including identifying information of “their” FL Training Collector.
4) Activated instances of the FL Local Operations establish connections with the FL Training Collector.
5) The FL Training Collector samples the FL Local Operations and provides them with model weights and, possibly, additional configuration, which triggers the training process on the FL Local Operations.
6) The FL Local Operations instances train the model (in parallel) and return the weights, along with any metrics they were requested to gather.
7) Next, the weights are aggregated according to a strategy supplied by the FL Training Collector. The data, along with requested metrics, is communicated (as required) before and after model evaluation (after each round).

This approach, formulated for the basic centralized architecture, can be then modified in order to support other topologies. Let us now describe how this can be achieved.
V. OTHER FL TOPOLOGIES, APPLICABILITY AND USABILITY

By slightly modifying the basic architecture, it is possible to instantiate other topologies, e.g. these proposed in the literature. In particular, four topologies have been implemented: centralized architecture, clustered architecture, hierarchical architecture, and star architecture with ring-based groups. They are illustrated in Fig. 3. It should be noted that the aim of this work was to establish that the proposed architectural approach, based on enablers originating from the ASSIST-IoT RA, can be used to easily set up “any” FL topology. Thus, this aspect of the architecture was implemented and tested. The utilization of these topologies for the car maintenance use case, described above, will be explored in the near future (including an evaluation of the FL process and the model performance).

The basic centralized architecture was implemented following the description presented above. The possibility of using different “parameters” of the FL process, as represented in the setup, including client numbers, model architecture, approach to model averaging, data collected by the FL Local Operations and FL Training Collector has been successfully tested.

As for the clustered architecture, the implemented version, first, accepts a number of clusters and then uses the Iterative Federated Clustering Algorithm (IFCA) [35] to dynamically determine the adherence of a given client to a cluster at the beginning of each round. Next, in the aggregation stage, the cluster models are updated, based only on the data from the clients that belong to them at a given moment. When faced with IID data, the clients are determined to belong to a single cluster, which leads the architecture to behave similarly to the centralized one. For the non-IID data, the clustered architecture allows for the development of a number of independent models, each tailored exactly to a given cluster of clients, instead of a single global solution.

The hierarchical topology requires the creation of an additional component [28]. Here, it was implemented as a special case of the FL Local Operations, called 1st Layer Local Operations. Therefore, the version of FL Local Operations acts as a basic FL client called 2nd Layer Local Operations.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
This (additional) enabler serves as an FL server to the 2nd Layer Local Operations and as an FL client to the FL Training Collector, aggregating updates from the 2nd Layer Local Operations during local training rounds, and propagating them to the global FL Training Collector for aggregation.

In the final experiment, the star topology with ring-based groups introduced decentralized elements based on the Tornadoes architecture [24]. Here, the training process starts with the FL Training Collector sending the initial model to all available FL Local Operations. Then, each of the FL Local Operations uses every local round to train the model on its local data to pass it on to the next instance belonging to its ring-based group, and accept an incoming model from the previous instance for further training. After a given number of local rounds, a global aggregation (performed by the FL Training Collector) occurs.

The implemented topologies were initially tested using CIFAR-10 (for IID data) and the German Traffic Sign Recognition Benchmark dataset (for non-IID data). Obtained results matched those found in [36], [37]. Note that the experiment description is out of the scope of this paper. Here, the focus is to show the applicability of the proposed architectures.

Finally, Fig. 4 presents a solution envisioned for the use case from Section II, using enablers from the proposed architecture.

Here, a centralized topology, where the FL Local Operations run on clients (cameras) is depicted. FL Orchestrator, FL Training Collector and FL Repository are located in the cloud. Note that due to the business nature of the use case, this environment is going to be somewhat more “stable” because of a specified number of clients (cameras) that are to participate in the process. The main goal is to distribute the processing, instead of sending all the images to the cloud and processing it centrally. Here, although the centralized topology seems to be a good choice for initial implementation, it is clear that a more complex topology will ultimately be needed. One of the reasons is that in extended deployment groups of scanners (one or more) may belong to different stakeholders. Therefore, a hierarchical topology would be a natural choice. Nonetheless, it has been already established (above) that such topology is easy to deliver using the existing set of enablers.

Note that, on the diagram, besides standard FL enablers, additional enablers designed and implemented within ASSIST-IoT (following ASSIST-IoT RA) are included addressing: Cybersecurity (specifically authentication and authorization), Long Term Storage (the Long Term Storage enabler can provide local storage of images for FL clients), and Tactile Dashboard (for visualizations needed in the system). Upon reflection, it is easy to see that these elements can provide all additional functions needed in the considered ecosystem.

VI. CONCLUDING REMARKS

Even though there is a lot of research in the field of Federated Learning, most of it is devoted to topics such as FL algorithms, processes, specific aspects of the processes such as data security, or how to deal with non-IID data. Here, we try to address issues related to the deployment of an FL system in a real-life use case, in an IoT ecosystem. This requires the choice of an appropriate architecture. In this context, the ASSIST-IoT RA was extended to deliver a set of enablers that allow for easy configuration of the FL system with machine learning parameters as well as any required topology. Moreover, additional enablers, created for the RA allow turning the FL process into a complete, robust solution.

As part of future work the aspect of adding human-in-the-loop will be addressed. The architecture should support scenarios, in which ground truth or feedback is required from the system user (an active-learning type scenario, with human acting as an Oracle), and is part of the FL process. On the other hand, the owner of an FL client should be able to set her preferences, e.g. with respect to acceptable resource usage. This can be foreseen as part of the system configuration and considered when selecting clients for the job. Moreover, experiments with selected models and described here topologies will be conducted to investigate their influence on the behaviour of the system and its learning process.

REFERENCES

[1] Q. Xia, W. Ye, Z. Tao, J. Wu, and Q. Li, “A survey of federated learning for edge computing: Research problems and solutions,” High-Confidence Computing, vol. 1, no. 1, p. 100008, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S266729522100009X
[2] Y. Roh, G. Heo, and S. E. Whang, “A survey on data collection for federated learning: A big data - ai integration perspective,” IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 4, pp. 1328–1347, 2021.
[3] B. Murdoch, “Privacy and artificial intelligence: Challenges for protecting health information in a new era,” BMC Medical Ethics, vol. 22, no. 1, 2021.
[4] H. Zheng, H. Hu, and Z. Han, “Preserving user privacy for machine learning: Local differential privacy or federated machine learning?” IEEE Intelligent Systems, vol. 35, no. 4, pp. 5–14, 2020.
[5] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, “A survey on security and privacy of federated learning.” Future Generation Computer Systems, vol. 115, pp. 619–640, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X20329848
[6] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, “Blockchain and federated learning for privacy-preserved data sharing in industrial iot.” IEEE Transactions on Industrial Informatics, vol. 16, no. 6, pp. 4177–4186, 2020.
[7] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, “Characterization and prediction of deep learning workloads in large-scale GPU datacenters,” CoRR, vol.abs/2109.01313, 2021. [Online]. Available: https://arxiv.org/abs/2109.01313
[8] X. Qiu, T. Parcollet, J. Fernández-Marqués, P. P. B. de Gusmão, D. J. Beutel, T. Topal, A. Mathur, and N. D. Lane, “A first look into the carbon footprint of federated learning,” CoRR, vol.abs/2102.07627, 2021. [Online]. Available: https://arxiv.org/abs/2102.07627
[9] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, “Federated learning of deep networks using model averaging,” CoRR, vol.abs/1602.05629, 2016. [Online]. Available: http://arxiv.org/abs/1602.05629
[10] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. Vincent Poor, “Federated learning for internet of things: A comprehensive survey,” IEEE Communications Surveys Tutorials, vol. 23, no. 3, pp. 1622–1658, 2021.
[11] S. Kumar, R. Schlegel, E. Rosnes, and A. G. i. Amat, “Coding for straggler mitigation in federated learning,” 2021.
[12] A. Tak and S. Cherkaoui, “Federated edge learning: Design issues and challenges,” IEEE Network, vol. 35, no. 2, pp. 252–258, 2021.
Fig. 4: FL architecture for the car damage use case