Data Hetero

Efficient Privacy-Preserving ML for IoT : Cluster-Based Split Federated Learning Scheme for Non-IID Data

Mohamad Arafeh*, Mohamad Wazzeh*, Hakima Ould-Slimane†, Chamseddine Talhi*, Azzam Mourad‡§, Hadi Otrok¶,
*Department of Software and IT engineering, Ecole de Technologie Superieure (ETS), Montreal, Canada
†Department of Mathematics and Computer Science, Universite de Quebec a Trois-Rivieres (UQTR), Canada.
§Cyber Security Systems and Applied AI Research Center, Department of CSM, Lebanese American University, Beirut, Lebanon
‡Division of Science, New York University, Abu Dhabi, UAE
¶Center of Cyber-Physical Systems (C2PS), Department of EECS, Khalifa University, Abu Dhabi, UAE

2023 7th Cyber Security in Networking Conference (CSNet) | 979-8-3503-4287-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSNET59123.2023.10339772

***Abstract*—This paper proposes a scheme addressing the challenges of integrating privacy-preserving distributed machine learning in the Internet of Things (IoT) context while improving the efficiency of the learning process and accelerating the convergence of the model. Specifically, it addresses the presence of non-independent and identically distributed (Non-IID) data and device heterogeneity, which occurs during parallel training and are common characteristics of IoT environments. In this context, SplitFed is a privacy-preserving technique that combines Federated Learning (FL) and Split Learning (SL) to enable fast and privacy-preserving model training on IoT devices. Through parallelism, FL allows fast and parallel model training. Furthermore, SL enables weak devices to train models through model splitting. However, SplitFed suffers from the increasing convergence time due to the FL parallelism introducing Non-IID and stragglers issues. In this paper, we aim to address the aforementioned problems by proposing a clustering parallelization scheme. As such, similar clients are combined based on their weight divergence and considered a single FL unit addressing the Non-IID problem. Moreover, to improve time efficiency and overcome the stragglers' presence, we offer a second clustering stage that separates clients by efficiency and uses FedAsync for model aggregation.**

***Index Terms*—Federated Learning, Non-IID, Privacy, Machine Learning, Participants Selection, Split Learning, Clustering**

I. Introduction

Nowadays, one of the main limitations when considering cybersecurity based on machine learning for IoT devices is their limited capabilities in terms of computational power and energy. One method to address this problem is to offload the training to centralized servers to train the models. In this case, IoT devices share raw data such as logs, traces, and binaries [7] with third-party servers, which filter and clean the data and proceed with the training process. However, this strategy involves participant sharing their sensitive data, which is a clear breach of their privacy. In addition, various organizations in different countries put restrictions on data collection, such as General Data Protection Regulation implemented by the European Union [6].

Recently, SplitFed [10] emerged as a solution to privacy-preserved distributed learning for IoT devices. SplitFed combines Federated Learning (FL) [5] and Split Learning (SL) [11], enabling fast and efficient model training. It split model weights into server and clients while allowing clients to train in parallel by assigning each to an external FL Server. When all clients finish training, the main server collects all clients and their server weights to aggregate the global model. It later sends it back, starting the next training round. Even though such an approach improved the training speed, it inherited all FL major issues, such as Non-IID and stragglers. Non-IID is highly emphasized in [13], [16], affecting the global model accuracy and increasing the time it needs to reach convergence. In addition, model aggregation usually occurs when all clients finish and send their weights to the server. Thus, clients who finish faster must wait for the rest of the clients to finish. In the context of IoT, the significant variety of devices' capabilities result in the presence of stragglers, substantially impacting the total time required to complete the task.

In this paper, we propose our approach as a solution to the Non-IID and stragglers problems. Our approach consists of two clustering phases: the out-cluster phase and the in-cluster phase. During the out-clustering phase, we tackle the Non-IID problem by clustering devices based on their Weight Divergence (WD) using their shared weights. WD has been introduced by [16] to identify the effects of involving Non-IID clients in federated learning approaches. According to [16], clients with a high weight divergence, introduced by their high Non-IID distribution, negatively affect the convergence time depending on their divergence level, which can be described following the approach in [1]. In such a context, WD can be used as a clustering input and will ensure that each cluster includes devices with a minimum WD. Aggregating such clients will have little to no effect on the cluster's global modal accuracy since they can be deemed IID combined.

During the second phase, the in-clustering, we address the stragglers problem by adopting an inner clustering approach. Within each out-cluster, we proceed with another clustering of clients based on their capabilities. By identifying the fastest cluster, we designate it as the primary cluster used for direct aggregation, while the rest queue their aggregated models to the out-cluster pool. Furthermore, we leverage a similar approach to FedAsync [15] to integrate queued models into the cluster's global model.

The main contributions are the following:

- We propose a clustering approach based on weight divergence to address the Non-IID problem. In this approach, the server group Non-IID clients into IID clusters. Furthermore, we run the inner-cluster clients in parallel using federated learning to speed up the training process. In contrast, the out-cluster runs in sequence to avoid the aggregation of biased models' weights.
- We propose a two-stage clustering approach to address the stragglers issue. The second stage considers clients' capabilities as input for clustering. Moreover, our approach integrates asynchronous capabilities to aggregate the late clusters' output.

II. Literature Work

Due to its ability to address privacy and security concerns, split learning has recently gained significant influence and is widely used by industry and academia. Additionally, SL opens opportunities to overcome challenging issues such as limited capabilities, bandwidth, and connectivity, making it especially useful in IoT environments. Numerous works have recently been published that improve SL capabilities to train ML models using IoT devices. For instance, the authors in [4] aims to improve the communication efficiency between clients and servers by introducing an auto-encoder that generates a sizable representation of the cut layer output.

However, despite the increasing interest, SL still needs to improve its time efficiency. As mentioned, SL only works sequentially due to its forced communication with the server. Additionally, SL's heavy reliance on clients' communication and computation capabilities highly affects its applicability in IoT, which mainly contains heterogeneous devices.

To address the aforementioned problem, most of the approaches in the IoT context integrate both SL and FL in a single solution. FL enables parallelization, while SL handles the device's computation and communication limitations. A combination of both approaches was first introduced by [10].

Recently, improved splitfed solutions have been introduced to address parallelism issues introduced by integrating FL, such as the stragglers' presence and device heterogeneity. The authors in [14] propose a clustering mechanism based on the clients' devices' properties to cluster devices with similar training times into the same group. Additionally, they propose a dynamic cut layer identification to reduce the training burden on devices with low computational power. Moreover, their main contribution lies in the workflow of their proposal, which makes only the devices within the same cluster run in parallel while the transmission from one cluster to another is sequential.

Although the mentioned proposal could handle the straggler issues, it did not focus mainly on targeting the issue of Non-IID distribution between clients that can significantly impact the convergence time when working in a federated learning environment.

III. Proposed Scheme

Figure 1 depicts our architecture. It comprises three key components: Main Server, Split Servers, and Clients. In the following, we describe each of the components.

- **Clients:** are the devices that will do the model training. The set of available devices is denoted by $C = \{1, 2, ...N\}$ where N denotes the total number of clients. Each client receives a partial model weight represented by wc, which is part of the whole model weights denoted by $w = \{wc; ws\}$ where ws is the rest of the model part trained by server. Finally, each client holds a local dataset $D_c = \{x_i, y_i\}$, where xi is a row of raw features input, while y_i represents the row label.
- **Split Server:** to ensure parallel model training, each client is assigned a dedicated SL server. Unlike traditional SL approaches, where clients depend on a single server for their model training, leading to synchronous communication and server lockouts. Following the proposed scheme, clients can train their models asynchronously without dependency on a shared server. We denote the set of Split Servers by $S = \{1, 2, ...N\}$ where $|S| = |C| = N$. Each Split Server holds its server weights w_s and its assigned client's weights w_c.
- **Main Server:** mainly communicate with the split servers. When the aggregation is due, the main server collects both weights $\{w_c; w_s\}$ from each split server included in the training and aggregates them to generate the new global models. The global models are then transferred to the next cluster assigned to the training.

Our approach combines a client c_i and split server s_i to train the model's weights $w = \{w_c; w_s\}$ asynchronously without sharing raw features with an outside entity (such as a split server or main server). Considering each client's dataset D_i^c, the main aim is to identify the optimal model parameters w_c by minimizing the loss function $\mathcal{L}(\mathbf{w}; \mathcal{D})$ defined by:

$$\mathcal{L}(\mathbf{w}; \mathcal{D}) = \sum_{i=1}^{|\mathcal{D}|} f(\mathbf{x}_i, y_i, \mathbf{w}) \quad (1)$$

When clients complete the training process with their respective split servers, the servers share the weights w with the main server. This will proceed with the aggregation according to the following:

$$w_{t+1} = \begin{cases} (1-\alpha)\mathbf{w}_t^{(i)} + \alpha \mathbf{w}_{t-s}^{(i)}, & \text{if queued} \\ \sum_{i=1}^{C} \frac{n_i}{n} w_{i,t+1}, & \text{otherwise} \end{cases} \quad (2)$$

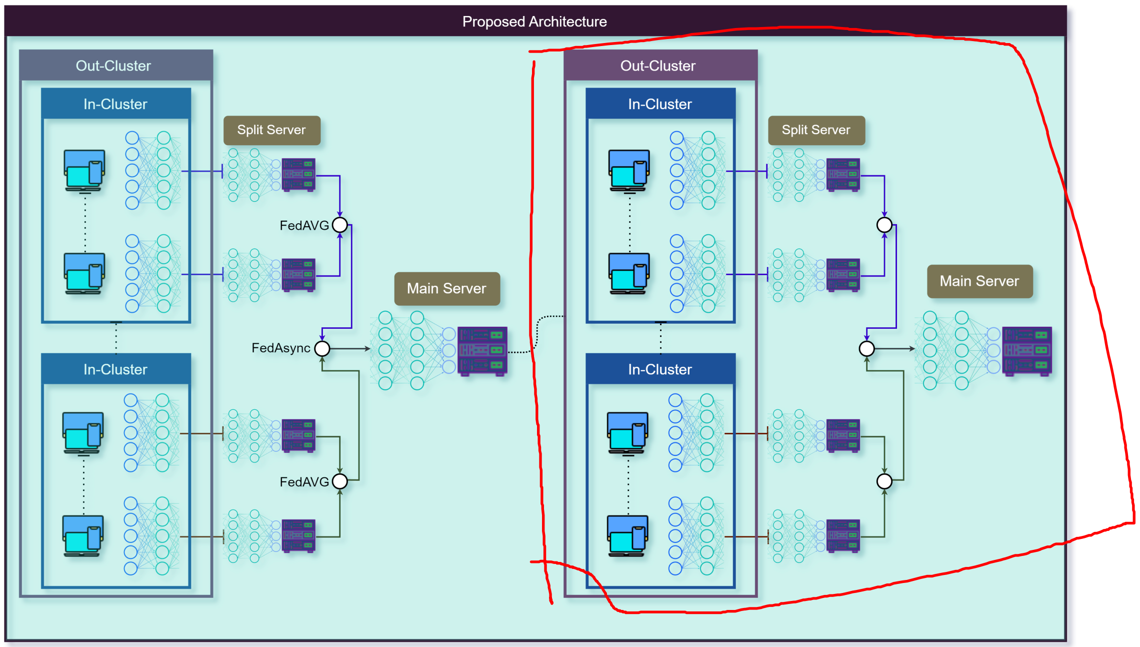

Fig. 1: Framework Architecture.

where s represents the stalled rounds, $\alpha = \frac{1}{s}$ represents the diluting factor, $\mathbf{w}_{t-\alpha}^{(i)}$ the stalled weights of cluster i, t latest round, n_i sample size of client i and n is the total sample size. As an example, if we suppose that all in-cluster finish at the same time, s in this case is equal to 1. Thus, we proceed with the normal aggregation method. In contrast, if one of the in-clusters was faster than the other, rather than waiting for the rest to finish, we proceed with the main cluster with normal aggregation while increasing s by one for the rest of the cluster and moving the out-cluster cursor sequentially to the next one selected randomly. In this case, if any of the previous out in-cluster finishes training, we proceed with the normal aggregation method for the local model and add the aggregate weights to the queue. When the aggregation cursor arrives at the out-cluster again, we aggregate the queued weight using the first part of the equation. As an example of queued integration, for $s_i = 2$ indicate that cluster i missed 3 aggregation rounds and its stalled weights $\mathbf{w}_{t-3}^{(i)}$ are diluted by a factor of 2, with $\alpha = \frac{1}{2}$, before aggregating them into the global model.

A. *Initialization*

During the initialization step, the server starts with a randomly generated model's weights and shares them with the clients. Each client trains its model with its assigned split server, which will send the results to the main server. In addition to the weights, the main server collects the clients' capabilities, such as computation speed, available energy, and network resources. Upon receiving all the required information, the server proceeds with the clustering using the KMeans algorithm [9].

The first clustering stage, the out-cluster, is based on weights divergence from the weights collected from clients during the initial stage. This approach ensures that the clients within the same cluster have minimal weight divergence and can be considered IID clients. In this case, cluster clients can work in parallel since aggregating their weights will not have a negative impact on the global model. In contrast, moving from one out-cluster to another occurs sequentially due to the significant weight divergence, which according to [16], has a high impact on the global model by increasing the convergence time and reducing the overall accuracy.

The second clustering stage splits the clients based on their capabilities to decrease the training latency. In the typical FL approach, in each round, clients that complete first must wait for the last clients to finish before the server aggregates their model and starts the next round. To overcome this issue, we build in-clusters to partition the clients with similar computation capacity and completion time. Moreover, using a similar approach FedAsync, the main server can integrate the delayed in-cluster aggregated weight to the global model by diluting them based on the stalled rounds denoted by l defined by the number of rounds the in-clusters devices missed the aggregation queue.

B. *Workflow*

In this section, we outline the design workflow of our approach in the following steps:

- Step 1 - Selection. The main server start by shuffling the deferred clusters and selects one randomly. The main server then distributes the global model to a predefined number of its split servers. The underlying mechanism behind the limited selection and random shuffling of clusters aims to avoid the catastrophic forgetting induced by integrating split learning [8] and as a replacement for the missing shuffling steps in the centralized mechanism.

- Step 2 - Client Training. In this step, the cluster's selected clients feed-forward their model to the specified cut layer in parallel, generating the smashed data and sending it to their assigned split server.
- Step 3 - Split Train. Split Servers infer $w_s(w_c)$ using their server model to generate the prediction result drawn from the data samples. Following this step, Split Server computes the loss and performs backward propagation up to the first server layer, which generates the gradients. These gradients are then sent back to the clients for them to continue the process on their layers.
- Step 4. Client Backward. In this step, each client used the received gradients to complete the backward propagation and update its optimizer. Processes 2-4 repeat in parallel for all in-cluster clients a number of times.
- Step 5. Aggregation. When at least one cluster's clients finish training, the assigned split servers send the weights to the main server, which will do the aggregation.
- Step 6. FedAsync. If the cluster queue contains aggregated weights from another cluster, the main server integrates them following the FedAsync as described in Equation 2. This step marks the end of the training with one out-cluster.
- Step 7. Transfer. The main server transfers the global model to another cluster selected randomly after reshuffling, repeating the same step 1-6. The main server completes one round when the global model is transferred to every available cluster.
- Step 8. Completion. The main server repeats steps 1-7 till it achieves the stopping criteria.

IV. Experimental Results

In this section, we evaluate the performance of our architectures compared to similar approaches. We conducted our experiments using Python following the ModularFed [2] library. In the following experiments, We compare our approach (C-SplitFed), to 3 others which are: SplitLearning (Split) [12], SplitFed [10], and SplitFed Clusters (Cluster) [14], presented in two benchmarks: accuracy and execution time. We used MNIST [1] to present our tests. The MNIST dataset contains images of handwritten digits between 0 and 9 distributed between 60k samples. These data are distributed to 100 clients, each with two labels [3].

Furthermore, our approach mainly focuses on improving the learning speed by bypassing the stragglers in IoT environments. Thus, it is essential to have heterogeneity in terms of capabilities in which it is possible to simulate multiple cluster speeds. Following this, we generate each IoT client with a speed control variable predefined based on the required performance relative to the host capabilities. Throughout our experiments, we set the number of clusters' speed to two: fast and slow clusters. The fast cluster can use all the capabilities of the host devices, while the slow cluster is set to a quarter of it. Additionally, we set our experiments capable of controlling the probabilities of slow and fast devices appearance based on parameter ψ used in uniform distribution random number generator. Through the remainder of our experiments, we fixed ψ to 0.4, indicating a 40% chance that the newly generated clients have slow capabilities.

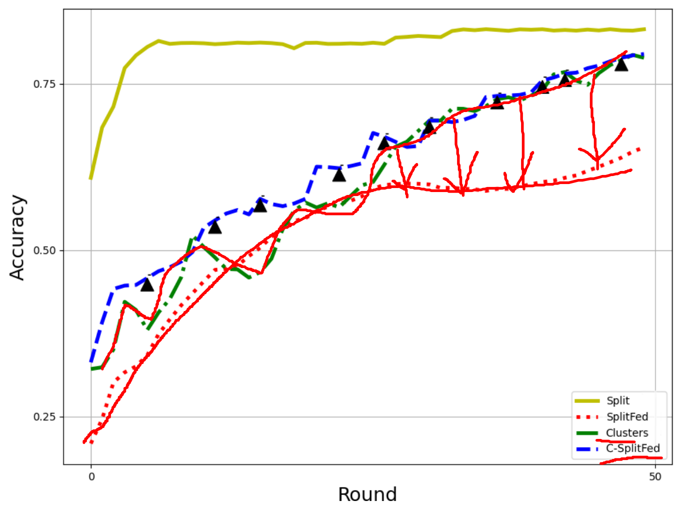

Fig. 2: Accuracy progress of our approach compared to Split, SplitFed, Cluster algorithm during the first 50 rounds on MNIST dataset with Non-IID distribution. The black arrowhead represents the occurrence of Async Aggregation.

Regarding the training model, our configuration consists of four linear layers, with the second layer considered a cut layer. The training hyper-parameters are fixed: rounds are 50, the batch size is 50, epochs are set to 1, and the server and the clients' learning rate is set to 0.001. We conducted our experiments only on a single epoch/round to simulate the limitation of the device's capabilities in the IoT environment.

Our first set of experiments is available in Figure 2, showing the evolution of accuracy in each round. Because aggregation is not needed, SplitLearning achieved superior results compared to the rest. Our approach and the clustering have achieved good final results when compared the SplitLearning, reaching a final accuracy of 0.794 compared to 0.822 for SplitFed. In contrast, SplitFed could only reach an accuracy of 0.656 due to the significant impact of aggregating clients with a high WD value. Such an issue is addressed in our approach, following that aggregation only occurs within each cluster which consists of clients that have reduced WD. Moreover, following our approach, we marked a black arrowhead every time we used FedAsync to integrate the queued weights of the slow clusters.

In our second set of experiments, we aim to evaluate the efficiency of our scheme when considering the stragglers. Figure 3 displays the total time each approach took to finish the 50 rounds, with Split taking the longest time due to its sequential workflow, finishing 50 rounds in almost 1200 seconds. In contrast, SplitFed achieves optimal execution time by adopting a fully parallel execution strategy. Compared to

[1]http://yann.lecun.com/exdb/mnist/

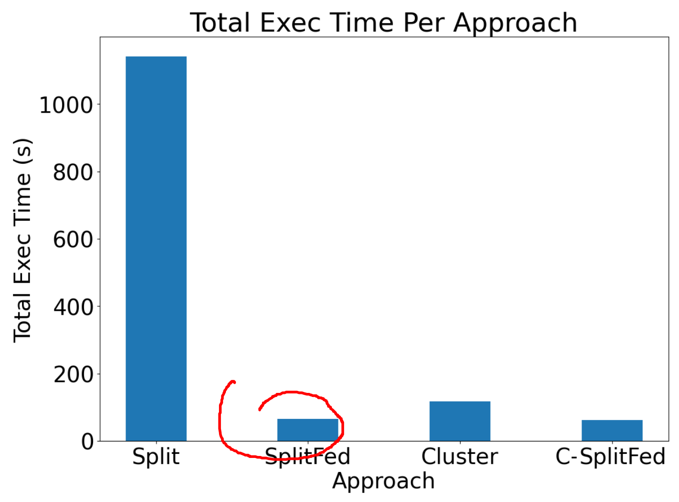

Fig. 3: Barchart showing the total execution time of each approach in seconds.

SplitFed, our approach achieved almost similar results, even though we are following a sequential cluster execution. This happens mainly due to the fact that Split, and SplitFed is highly affected by the existence of stragglers. Using our approach, we were capable of bypassing these stragglers by placing them in another cluster, making our approach as fast as SplitFed, which is considered the baseline approach we use for comparison.

V. Conclusion

In this paper, we discuss the impact of Non-IID distribution and the heterogeneity of clients' devices on the integration of the federated split learning approach in securing IoT devices. To tackle these challenges, we propose a novel double-stage clustering approach. Our proposed approach consists of two clustering stages: the out-cluster and the in-cluster. The out-cluster stage addresses the issue of Non-IID data distribution among clients by grouping them based on their weight divergence. In the in-cluster stage, clients within the out-cluster groups are further grouped based on their execution time to mitigate the impact of stragglers in the network. Our proposed workflow comprises Async parallel in-cluster training followed by a sequential out-cluster model transfer.

Our experimental results demonstrate the effectiveness of our approach in achieving efficient execution times and improved convergence compared to other approaches.

Acknowledgement

This work was supported in part by funding from the Innovation for Defence Excellence and Security (IDEaS) program from the Department of National Defence (DND)

References

[1] Mohamad Arafeh, Ahmad Hammoud, Hadi Otrok, Azzam Mourad, Chamseddine Talhi, and Zbigniew Dziong. Independent and identically distributed (iid) data assessment in federated learning. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 293–298, 2022.

[2] Mohamad Arafeh, Hadi Otrok, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Ernesto Damiani. Modularfed: Leveraging modularity in federated learning frameworks. *Internet of Things*, 22:100694, 2023.

[3] Mohamad Arafeh, Hakima Ould-Slimane, Hadi Otrok, Azzam Mourad, Chamseddine Talhi, and Ernesto Damiani. Data independent warmup scheme for non-iid federated learning. *Information Sciences*, 623:342–360, 2023.

[4] Ahmad Ayad, Melvin Renner, and Anke Schmeink. Improving the communication and computation efficiency of split learning for iot applications. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 01–06, 2021.

[5] Mingzhe Chen, Nir Shlezinger, H. Vincent Poor, Yonina C. Eldar, and Shuguang Cui. Communication-efficient federated learning. *Proceedings of the National Academy of Sciences*, 118(17):e2024789118, 2021.

[6] European Commission. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance), 2016.

[7] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.

[8] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2015.

[9] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[10] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8485–8493, 2022.

[11] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[12] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[13] Mohamad Wazzeh, Hakima Ould-Slimane, Chamseddine Talhi, Azzam Mourad, and Mohsen Guizani. Warmup and transfer knowledge-based federated learning approach for iot continuous authentication. *arXiv preprint arXiv:2211.05662*, 2022.

[14] Wen Wu, Mushu Li, Kaige Qu, Conghao Zhou, Xuemin Shen, Weihua Zhuang, Xu Li, and Weisen Shi. Split learning over wireless networks: Parallel design and resource management. *IEEE Journal on Selected Areas in Communications*, 41(4):1051–1066, 2023.

[15] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[16] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.