Mitigating System Bias in Resource Constrained Asynchronous Federated Learning Systems

Jikun Gao*, Ioannis Mavromatis†, Peizheng Li*, Pietro Carnelli*, Aftab Khan*

*Bristol Research and Innovation Laboratory, Toshiba Europe Ltd., U.K.

†Digital Catapult, London, U.K.

Email: {Peizheng.Li, Pietro.Carnelli, Aftab.Khan}@toshiba-bril.com, Ioannis.Mavromatis@digicatapult.org.uk

***Abstract*—Federated learning (FL) systems face performance challenges in dealing with heterogeneous devices and non-identically distributed data across clients. We propose a dynamic global model aggregation method within Asynchronous Federated Learning (AFL) deployments to address these issues. Our aggregation method scores and adjusts the weighting of client model updates based on their upload frequency to accommodate differences in device capabilities. Additionally, we also immediately provide an updated global model to clients after they upload their local models to reduce idle time and improve training efficiency. We evaluate our approach within an AFL deployment consisting of 10 simulated clients with heterogeneous compute constraints and non-IID data. The simulation results, using the Fashion-MNIST dataset, demonstrate over 10% and 19% improvement in global model accuracy compared to state-of-the-art methods PAPAYA and FedAsync, respectively. Our dynamic aggregation method allows reliable global model training despite limiting client resources and statistical data heterogeneity. This improves robustness and scalability for real-world FL deployments.**

***Index Terms*—Machine Learning, Federated Learning, Scalability, Resource-constrained Devices, System Bias, Device Heterogeneity.**

I. Introduction

Federated Learning (FL) is a collaborative Machine Learning (ML) method, which aims to meet the significant challenges of (user) data privacy and the large number of devices. FL was first proposed by Google/Alphabet in 2016 [1]. In traditional ML paradigms, datasets are centralised in a single location for processing and learning [2]. Whereas, with an FL framework, data is distributed on different devices and no longer needs to be collected centrally. Each participating device (also known as a *"client"*) will train the model independently using their local data. Once a training round is complete (usually involving several learning iterations), the client sends the gradient or parameter update of the model to a central aggregation node, known as the *"parameter server"* (PS). The task of the PS is to aggregate the updates sent by all the involved clients, form a new global model, and broadcast the new global model back to each client [3]. FL framework ensures that user data never leaves the original device, and only model parameters are shared centrally, thus protecting user privacy to a certain extent [4].

In practical FL implementations, participating client devices exhibit heterogeneity in hardware capabilities, including communication link quality, bandwidth availability, local memory capacity, processing speed, and power constraints. In addition, given the nature of distributed devices and sensors in an FL network, the data held by participating FL devices typically follows a non-Independent and Identically Distribution (non-IID) [5]. Often these differences in hardware performance and training data distribution lead to a critical problem in real-world FL applications: how can we process the time difference in completing a round of FL training between different clients [6]. In the synchronised version of FL [7], more efficient (in terms of computational capacity) devices are often forced to wait after completing their local training tasks. This can be mainly attributed to the FL devices with inferior hardware capabilities or those with larger local datasets requiring substantial computing resources to complete client-side training procedures. Only after these devices have successfully shared their locally trained models with the parameter server can the more efficient devices proceed with the subsequent round of training [8]. This means that the efficiency of the entire system can be negatively impacted by the worst-performing client, resulting in a large number of computing resources being wasted/idling unnecessarily [9]. The distribution of non-IID data can also impede global model convergence during collaborative training as each client model drifts towards disparate local optima [10]. Existing mitigation techniques such as aggregation after receiving a pre-set minimum number of local models [11] carry inherent limitations; slower-updating edge devices face exclusion from

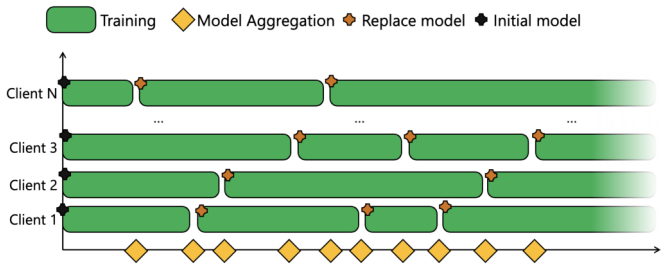

Fig. 1. A typical AFL system overview. Starting from the left, an initialised model is sent to all clients. Clients commence training (green), and once completed, they share their model with the parameter server. Once a new client model is uploaded, it is immediately aggregated into the global model (yellow diamonds) and sent back to the client.

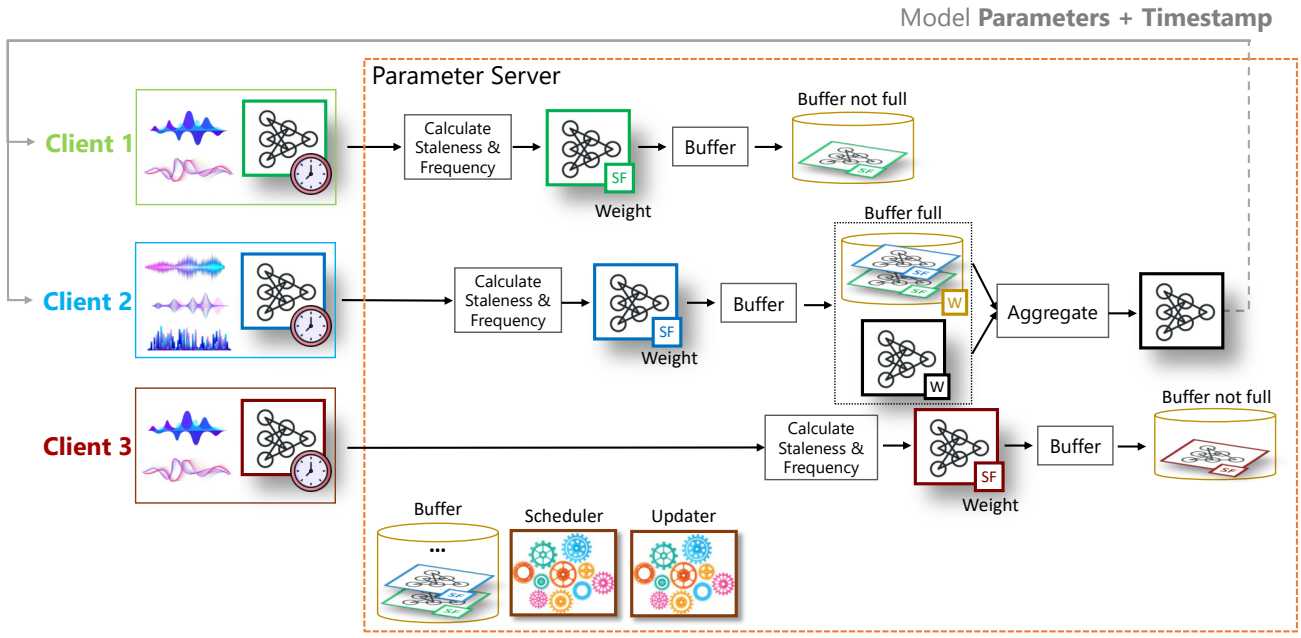

Fig. 2. Our asynchronous FL system overview diagram. Clients train using local datasets before sharing model parameters with the Parameter Server. In turn, client models are scored/weighted according to the frequency of updates before being aggregated into a new global model for sharing with client devices.

contributions to the global model aggregation regardless of representativeness.

In order to solve the problems of delay and resource utilisation in such traditional FL settings, Asynchronous Federated Learning (AFL) was proposed (illustrated in Fig. 1). In this, the parameter server no longer waits for all devices or edge nodes to complete their respective training [12] prior to broadcasting a global model. Once the device completes training of its local model, it is shared with the parameter server. The server then immediately starts aggregating the new local model with the existing global model and sends the updated global model back to the device so that the device can continue the next round of training. This aggregation process also usually involves combining models from different devices. In most cases, these models are given equal weighting (when averaged) to generate a new global model [13]. The advantage of this method lies in the high tolerance towards devices with different hardware capabilities and data distribution. Especially in the case of significant differences in device resources, the AFL strategy ensures that devices with superior computing performance will not waste time waiting, thus avoiding a large number of computing resources in the idle state [14]. Since, each FL device can train a model and upload it independently, the communication or waiting with other devices is significantly reduced. This method has significant advantages in communication efficiency, especially in those scenarios with high network delay or unstable network connection [15].

While AFL addresses challenges related to the operational functionality of FL deployments, the global model remains susceptible to bias from hardware heterogeneity and non-IID data distributions. Consequently, devices updating the global model more frequently may dominate aggregation, creating a skew that disadvantages resource-constrained edge devices with infrequent updates. Additionally, if not appropriately weighted for recency, stale model updates can disrupt representation. Similarly, non-IID data partitioning and class imbalance can lead to the training of disparate edge models. Naively aggregating these models may hinder convergence, resulting in suboptimal global model performance. Therefore, intelligent coordination in AFL aggregation is essential to ensure robustness against device and data heterogeneity, preventing compromised global model accuracy.

This paper introduces an enhanced AFL model aggregation method that evaluates and adjusts global model aggregation based on client upload frequency. Furthermore, it actively reduces client idling time between FL aggregation rounds by providing clients with the latest global model for training. During the FL client upload phase, a buffer layer is implemented, incorporating an aggregation scaling factor to quantify communication frequency with the server. It calculates the difference between the client model being uploaded and the global model. Dynamic parameters within the layer are configured to optimise the induced model staleness bias. Consequently, our method can train a reliable global model even when faced with limitations in the computing power of an FL client or the uneven distribution of local client training

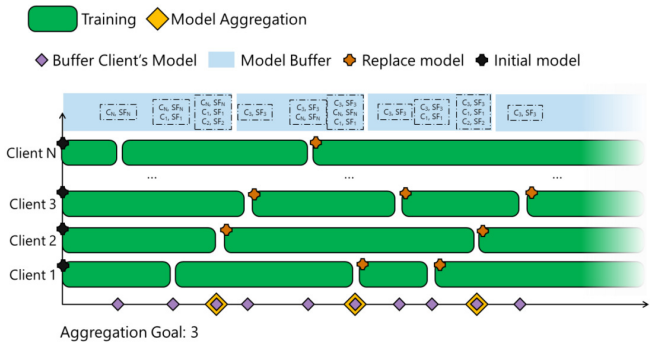

Fig. 3. Diagram of our proposed method with an aggregation goal of 3 client models. Clients immediately receive the latest aggregation/global model once they upload their local model to the PS. However, we store and score the incoming client models in the PS buffer layer to reduce biases when aggregated. If the new global model is sufficiently different (once the buffer is full) it is then re-shared with all the clients.

data, addressing both data volume and class imbalance.

In the subsequent sections, the paper is structured as follows: Section II covers the background to our method detailed in Section III. We discuss our experimental setup in Section IV. Section V shows our results which are further discussed and concluded in Section VI.

II. Related Work

In order to solve the potential bias in AFL environments, the FedAsync algorithm, proposed by Xie *et al.* [12], introduces a dynamic hybrid hyperparameter based on an outdated model. For example, when the current global model has been updated for a few rounds of aggregation and a new client has completed the first delayed local update, outdated or non-obsolete models will be queued in chronological order. The PS controls the proportion of local models when aggregating with the global model through obsolescence functions. Consequently, local models with a large degree of 'staleness' will have a small weighting in the global model update. This algorithm directly reduces the impact of the stale/laggard model on the accuracy of the global model. However, these models may still contain some valuable information that could help to improve the accuracy of the global model.

Huba *et al.*'s Papaya algorithm [16] implements the FedBuffer algorithm proposed by Nguyen *et al.* [17]. In this, after a client completes its local training, the updated local model is uploaded to the buffer layer memory first. When the buffer layer reaches the aggregation target (i.e., a pre-set number of client models), the global model is aggregated and updated to the clients. This can effectively reduce the gap between the laggard and the faster clients, but the computing resources of some powerful devices are still not optimally utilised because these clients still need to wait for the weaker devices in the buffer layer to complete the training and enter the idle state.

At present, to the best of our knowledge, there is no method that can reduce the impact of aggregating stale models with the global model accuracy whilst also optimising for the computing resources of FL clients. In this paper, we propose a method to mitigate the impact of client hardware and data heterogeneities on the global model.

III. Methodology

TABLE I
Table of notation.

Notation	Description
N	Total number of clients
$[n]$	Set of integers $\{1, \ldots, n\}$, to represent each client
T	Total number of global communication rounds
t	Timestamp of the current number of global communication rounds
τ	Timestamp of the current local model based on which round of global model
x_t	Global model in the tth global communication round on server
D_n	Dataset on the nth device
α	Global model scale factor
β_n	Intra-group model scale factor of nth client
B	Total number of intra-group clients, equal to the buffer aggregation target
$[b]$	Set of integers $\{1, \ldots, b\}$, to represent each intragroup client
$t - \tau$	Directly obtained staleness
s	Function of staleness
f	Model upload frequency

Our proposed framework for mitigating system bias, as introduced earlier, follows the asynchronous FL setup. A system overview diagram is depicted in Fig. 2 and Fig. 3 illustrates some example training rounds of our method. When the buffer layer at the PS receives a new local model from the participating FL client devices, and the buffer layer does not meet the aggregation target (i.e., the minimum number of client models in the buffer layer, prior to triggering a global model aggregation task), it immediately shares the current global model with the client device to continue the training task. Upon reaching the aggregation target, the buffer layer promptly aggregates the new global model, replacing the previous global model. This ensures the maintenance of a unique global model throughout the FL training process

When any client device finishes uploading its locally trained model to the buffer layer, it will no longer enter the idle state but immediately obtain the latest global model for continued local training (thus avoiding computing resources idling/waiting). However, in this case, client devices with more compute resources may upload more than once in the same buffer layer (i.e., within a short time frame). Consequently, we introduced a scaling factor $\beta \in [0, 1]$ in the buffer layer to balance the impact of client delays caused by device heterogeneity or local data heterogeneity.

Our proposed aggregation strategy is outlined in Algorithm 1, and a comprehensive list of notations used in this paper is listed in Table III. Upon the receipt of the local model at the PS, we evaluate the obsolescence/staleness of the current model based on the disparity between its and the latest global model's timestamps – greater disparities signify a "stale" local client model. Moreover, when added to the buffer, stale models are assigned a smaller scaling factor for

Algorithm 1 Proposed method - Dynamic weight buffered

procedure SERVER($\alpha \in (0, 1)$)
  Initialise $x_0, x_{new}, \alpha, b \leftarrow 0, B, t \leftarrow 0, T$
  Run SCHEDULER() thread and UPDATER() thread asynchronously in parallel
end procedure
procedure SCHEDULER
  Periodically trigger training tasks on all clients and send the global model with a timestamp
end procedure
procedure UPDATER
  for $t \in [T]$ **do**
    Receive the pair (x_{new}, τ) from any client,buffered
    $x_b \leftarrow$ buffered x_{new}
    $\beta_b \leftarrow$ Calculate the intragroup scale factor corresponding to each client staleness and upload frequency
    $x_{\text{new}}(t) \leftarrow x_{\text{new}}(t) + \beta_b * x_b$
    $b \leftarrow b + 1$
    if $b == B$ **then**
      $x_t \leftarrow (1 - \alpha) * x_{t-1} + \alpha * x_{\text{new(t)}}$
      $b \leftarrow 0$
      $x_{\text{new}}(t) \leftarrow 0$
    else
      $x_t \leftarrow x_{t-1}$
    end if
    Send x_t to relative client
    $t \leftarrow t + 1$
  end for
end procedure
procedure CLIENT
  for $n \in [N]$ in parallel **do**
    if triggered by the scheduler **then**
      Receive the pair of the global model and its timestamp (x_t, t) from the server
      $\tau \leftarrow t, x_n \leftarrow x_t$
      Train based on the local dataset and upload after training is completed
    end if
  end for
end procedure

aggregation. In the case of different local models from the same client being received before reaching the aggregation target in the buffer, a smaller scaling factor will be assigned to the particular high frequency client. This adjustment aims to prevent performance skewness towards resource-efficient clients, which could otherwise lead to an underrepresented global model. When the aggregation target is reached, we aggregate the sum of the scaled local models currently stored in the buffer as a new global aggregation model. The scale factor β of the n-th client within the buffer can be defined as:

$$\beta_n = \frac{|D_n| \cdot e^{s_n \cdot \frac{1}{f_n}}}{\sum_{b=1}^{B} |D_b| \cdot e^{s_b \cdot \frac{1}{f_b}}} \tag{1}$$

TABLE II
COMMON HYPERPARAMETERS SETTINGS FOR OUR SIMULATED AFL EXPERIMENTS.

Parameter	Value
Global communication rounds	100
Scale Factor (α)	0.5
Buffer Aggregation Target	3
Learning Rate (η)	0.001
Regularization Parameter (μ)	0.01
Local Training Epoch (E)	2
Local Minibatches (b)	50
Model for Training Task	CNN
Optimizer for Training Task	SGD

where:

- $s = (t - \tau + 1)^{-\alpha}$ represents the model's staleness factor within the buffer layer.
- f represents the model's upload frequency within the buffer layer.
- B is the total number of devices in the buffer layer.
- $|D_n|$ represents the data samples stored on the n^{th} device.
- $\sum_{b=1}^{B} |D_b|$ is the total number of data samples stored across all B devices in the buffer layer.

IV. EXPERIMENTS

To evaluate the proposed aggregation strategy (detailed above) within AFL settings, we employ the clothing-based Fashion-MNIST image recognition task for benchmarking [18]. This dataset contains 60,000 training and 10,000 test images, formatted as 28×28 grayscale images spanning 10 apparel classes. We distribute this training data in both IID and non-IID setups across the local datasets of 10 simulated FL clients and then quantify model performance aggregated from their intermittent updates. We compare our dynamic, latency-adjusted aggregation scheme against two state-of-the-art approaches, as introduced in Section II: FedAsync [12] and Papaya [16].

In order to assess the most salient bias in AFL (considering the FedAsync method) and its impact on the overall accuracy of the global model. We designed an experimental setup consisting of 10 FL clients (summarised in Table III). To mimic hardware disparities, half of the clients were deliberately subjected to resource limitations by introducing delays in their updates. Additionally, we investigated the consequences of data distortion or class imbalance of local client training datasets by varying the number of classes (out of a total of 10) for each client during local training. Finally, an IID experiment, where classes and training data volumes were uniform across all FL clients, was also conducted to illustrate the "accuracy cost" associated with such modifications. The hyperparameters used for all simulated AFL experiments are consistent and detailed in Table II.

We designate the simulated FL clients as $c_0 - c_9$, and their computing capabilities correspond to the following fractions $[100, 95, 90, 85, 80, 25, 20, 15, 10, 5]$, where the larger values implied better computing resources available. The Dirichlet

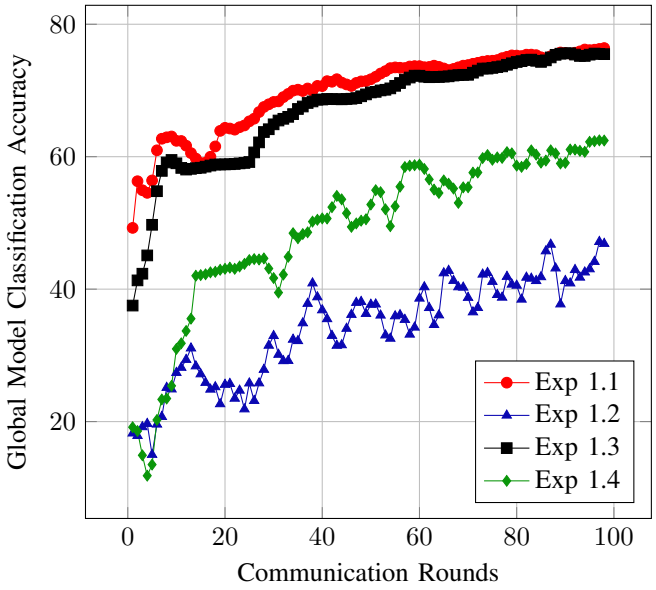

Fig. 4. Results plotted for IID versus various Non-IID FL client training dataset distributions settings with their computing resources corresponding to the following units/fractions $[100, 95, 90, 85, 80, 25, 20, 15, 10, 5]$ where the larger values imply better computing resources. Note, that the hyperparameter settings used are shown in Table II. Exp 1.1: IID client training data, Exp 1.2: Resource constrained devices with 3 training classes, Exp 1.3 Resource constrained devices with all training classes, and Exp 1.4: Client training data with Dirichlet distribution.

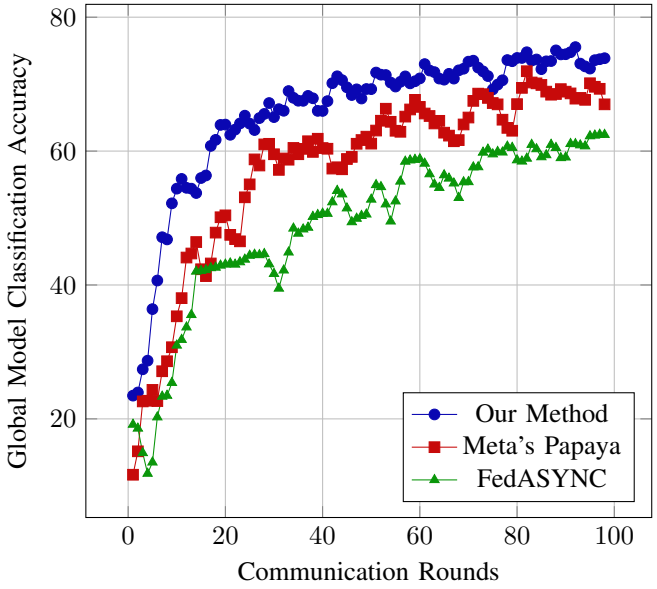

Fig. 5. Comparison between our proposed method, Meta's PAPAYA [16] and FedAsync [12] AFL systems illustrating classification accuracy using the FashionMNIST dataset under experimental setup 1.4 as described Table III, and hyperparameter settings shown in Table II.

TABLE III
EXPERIMENTAL HYPER-PARAMETER SETUP FOR RESULTS IN FIGURE V

Experiment ID	FL client compute power	data split	Class distribution per client
1.1	50% resource constrained	IID	10
1.2	50% resource constrained 50% resource efficient	non-IID	10 3
1.3	50% resource constrained 50% resource efficient	non-IID	3 10
1.4	50% resource constrained	non-IID	Dirichlet

distribution was employed to generate the number of classes available per client to train within the final row (Exp. 1.4).

In our second experimental setup, we evaluated the differences in performance of the global accuracy during AFL training of our proposed method compared with other advanced algorithms such as FedAsync [12] and Papaya [16]. We used the same experimental settings as the final row (Exp. num. 1.4) of Table III and hyperparameters from Table II.

V. RESULTS

From Fig. IV, it is evident that, in Experiment 1.1, the heterogeneous training devices based on IID data achieve the highest global model accuracy. Experiment 1.2 represents computationally resource constrained clients training with access to all 10 classes of the FashionMNIST dataset and the computationally efficient clients training with only 3 classes can achieve almost similar results as the IID setting. However, in the reversed scenario (Experiment 1.3), where weaker clients' training is based on all 10 classes and stronger clients train on only 3 classes, this results in the worst performance in terms of classification accuracy.

Experiment 1.4 used the Dirichlet distribution setting, wherein each client was assigned on average 4 or 5 classes of training data (out of a possible 10). This choice was made to emulate a more practical and diverse FL setup, ensuring that no single client had access to the full set of classes to train with. Across these four different experimental settings, it is evident that more powerful clients exert a dominant influence on the training process, and variations in computing resources and training data do introduce bias in the global model. Performance deterioration is particularly noteworthy in scenarios where resource constrained clients have access to the majority classes of the dataset. This can be attributed to the fact that slower clients, with a reduced update frequency, have a lesser contribution to the global model which is heavily biased towards clients with the most amount of updates.

We next evaluated our proposed method targeting mitigation against such bias and compared its performance with two state-of-the-art AFL methods in PAPAYA and FedAsync. Global model accuracy for each during AFL training is shown in Fig. V. We used the simulated experimental setup as described in row 1.4 of Table III. The results show that our proposed method improves the FashinMNIST classification performance by more than 10% and 19% when compared to the PAPAYA and FedAsync methods respectively (for a fixed set of communication rounds).

VI. Discussion and Conclusion

In this paper, we conducted multiple experiments based on an Asynchronous FL environment to evaluate the biases caused by FL device hardware heterogeneity and non-IID client data distributions. High-performance devices in these settings quickly train and update models in asynchronous environments, while resource constrained devices may delay submitting their updates. This mismatch can lead to under-consideration of updates for some devices when aggregating global models at the PS, as they are relatively outdated/stale compared to the updated global model state. In addition, network latency and interruptions may also lead to the loss or delay of certain device updates, which poses a threat to the stability and convergence of the global model.

Moreover, data heterogeneity also negatively impacts performance in such FL applications. The training data stored in the FL client device may vary significantly due to geographical location, user behaviour habits, or other factors. This means that the data of certain devices may not represent the overall data distribution (of all FL clients), or some key subsets of data may only exist on a few devices. Such data distribution may lead to excessive optimisation of the local model on specific tasks or subsets of data, thereby compromising the global ability to generalise to new data or different devices. In terms of data volume, some devices may have rich data (lots of classes, large volume), while others may have relatively few. This imbalance, without proper processing, can cause data-rich devices to have an overwhelming impact on the global model.

In this work, we show through extensive experimentation that such bias, caused by hardware and data heterogeneity, in AFL, cannot be completely eliminated but can be appropriately reduced. We have proposed a new aggregation method to reduce the impact of such bias on the global model performance. Our method has been experimentally demonstrated to effectively improve the global model accuracy in AFL environments when compared with other state-of-the-art benchmark algorithms.

For future work, we aim to cover other datasets and different model architectures to evaluate the robustness of our method. Moving forward, promising avenues exist to expand the evaluation of our aggregation technique across even more heterogeneous AFL system deployments. Additional dimensions of hardware diversity could entail sampling various device computation, memory, and power constraints. On the model representation side, assessing performance under conditions of quantisation or pruned neural networks tailored to match the resource limitations of local devices would prove highly valuable. Developing robust, adaptive AFL aggregation schemes capable of harmonising diverse hardware configurations, statistical data divergences, and specialised local model representations remains an open challenge of great practical interest.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[2] F. Naeem, M. Tariq, and H. V. Poor, "Sdn-enabled energy-efficient routing optimization framework for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5660–5667, 2020.

[3] L. Ferraguig, Y. Djebrouni, S. Bouchenak, and V. Marangozova, "Survey of bias mitigation in federated learning," in *Conférence francophone d'informatique en Parallélisme, Architecture et Système*, 2021.

[4] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[5] Y. Li, S. Yang, X. Ren, and C. Zhao, "Asynchronous federated learning with differential privacy for edge intelligence," *arXiv preprint arXiv:1912.07902*, 2019.

[6] E. Diao, J. Ding, and V. Tarokh, "Heterofl: Computation and communication efficient federated learning for heterogeneous clients," *arXiv preprint arXiv:2010.01264*, 2020.

[7] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous federated learning on heterogeneous devices: A survey," *arXiv preprint arXiv:2109.04269*, 2021.

[8] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[9] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[10] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[11] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[12] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[13] M. Chen, B. Mao, and T. Ma, "Fedsa: A staleness-aware asynchronous federated learning algorithm with non-iid data," *Future Generation Computer Systems*, vol. 120, pp. 1–12, 2021.

[14] Z. Chen, W. Liao, K. Hua, C. Lu, and W. Yu, "Towards asynchronous federated learning for heterogeneous edge-powered internet of things," *Digital Communications and Networks*, vol. 7, no. 3, pp. 317–326, 2021.

[15] M. Chen, B. Mao, and T. Ma, "Efficient and robust asynchronous federated learning with stragglers," in *International Conference on Learning Representations*, 2019.

[16] D. Huba, J. Nguyen, K. Malik, R. Zhu, M. Rabbat, A. Yousefpour, C.-J. Wu, H. Zhan, P. Ustinov, H. Srinivas *et al.*, "Papaya: Practical, private, and scalable federated learning," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 814–832, 2022.

[17] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 3581–3607.

[18] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.