2023 IEEE 41st International Conference on Computer Design (ICCD) | 979-8-3503-4291-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCD58817.2023.00074

FLASH-RL: Federated Learning Addressing System and Static Heterogeneity using Reinforcement Learning

Sofiane Bouaziz*, Hadjer Benmeziane†, Youcef Imine†, Leila Hamdad*, Smail Niar†, Hamza Ouarnoughi†
* École nationale Supérieure d'Informatique, Algiers, Algeria
†Univ. Polytechnique Hauts-de-France, CNRS, UMR 8201 - LAMIH, F-59313 Valenciennes, France

***Abstract*—We propose FLASH-RL, a framework utilizing Double Deep Q-Learning (DDQL) to address system and static heterogeneity in Federated Learning (FL). FLASH-RL introduces a new reputation-based utility function to evaluate client contributions based on their current and past performances. Additionally, an adapted DDQL algorithm is proposed to expedite the learning process. Experimental results on MNIST and CIFAR-10 datasets demonstrate that FLASH-RL strikes a balance between model performance and end-to-end latency, reducing latency by up to 24.83% compared to FedAVG and 24.67% compared to FAVOR. It also reduces training rounds by up to 60.44% compared to FedAVG and 76% compared to FAVOR. Similar improvements are observed on the MobiAct Dataset for fall detection, underscoring the real-world applicability of our approach.**

***Index Terms*—Federated Learning, Reinforcement Learning, System heterogeneity, Static heterogeneity, Client selection.**

I. Introduction

FL is a promising approach to Machine Learning (ML), offering privacy-preserving model training in distributed networks. It enables edge devices to collaboratively train a shared ML model while maintaining control of data on the edge level. In FL, each participant independently trains a model on its local data and shares the updated model parameters with a central server for aggregation. Existing works [1]–[3] primarily focus on optimizing this aggregation process. In real-world scenarios, where the number of clients is large, selecting the right subset of clients for each training round is crucial, impacting convergence and model performance.

A key challenge in FL is the presence of *system* and *static* heterogeneity across the participating clients, described in section II-A. Several works [4] address client selection to tackle system and static heterogeneity. They aim to choose the best clients that represent the entire client distribution while also considering their computational resources. However, most studies focus on one type of heterogeneity, lacking a unified approach.

This paper introduces a novel and efficient FL client selection strategy using reinforcement learning (RL), to address both system and static heterogeneity. The contributions presented in this paper are as follows:

- We define a reputation-based utility function that assesses the performance of each client's local model during training and the latency incurred in their training.
- We design double deep Q-learning (DDQL) and multi-action selection-based RL algorithms to perform the client's selection during the FL process.
- We apply our approach in a real-world use-case which is fall detection in smart homes.

The rest of the paper is organized as follows. Section II covers FL and related works. Section III-A explains latency calculation. Section III-B defines the reputation-based utility function. Section III-C outlines client selection based on RL. Section IV details the experimental methodology and results.

II. Background & Related Works

A. Federated Learning & Heterogeneity problems

In FL [5], we consider N clients with individual datasets. The FL optimization problem is described as follows in Equation 1, where F_k is the local objective function, $p_k > 0$ and $\sum_{i=1}^{N} p_k = 1$.

$$\min_{w \in R^d} \phi(w) = \sum_{k=1}^{N} p_k F_k(w) \tag{1}$$

In each training round, the server chooses a subset of clients to prevent network overload. This client selection needs to account for FL heterogeneity problems [6]:

- **Static heterogeneity**: Client data is often distributed non-identically and non-independently (non-IID), resulting in heterogeneous data distribution between clients.
- **System heterogeneity**: The storage, computing, and communication capacities of each client may differ due to the variability of hardware, and network connectivity. This leads to variability in the transfer time of each client.

Despite the successes of existing FL algorithms [5], [7], they still exhibit instability in highly heterogeneous environments.

B. Reinforcement Learning-based Solutions

Recent studies [8]–[11] demonstrate the potential of RL-based client selection in FL. The latter proposes an optimization framework that enables adaptive client selection by considering data distribution and hardware resources. In [8], they introduce FAVOR which uses DDQL to handle static heterogeneity but relies on a single client for DDQL training, slowing agent convergence. Similarly, [9] and [10] use DDQL for client selection, focusing on system heterogeneity. In [11],

2576-6996/23/$31.00 ©2023 IEEE
DOI 10.1109/ICCD58817.2023.00074

they introduce a multi-agent RL (MARL) system to handle system and static heterogeneity in client selection. However, this approach's use of multiple agents increases resource demands, making it resource-intensive.

C. Fall Detection Use-case

Fall detection is a classification task that draw on sensor data to distinguish falls from Activities of Daily Living (ADL). In centralized solutions, [12] employs Long Short-Term Memory for ADL classification. In [13], ensemble methods are used to classify data into ADL or fall. FL-based solutions feature the work of [14], which presents *FedHome*, a framework that enables fall detection while preserving data privacy.

III. Proposed Approach

This section describes the objective functions employed to account for system and static heterogeneity. We then propose a DDQL-based solution that integrates them into the reward function for client selection.

A. Handling System Heterogeneity: Latency

We consider latency as the objective function to handle system heterogeneity. It's defined for a selected client k at training round t as the sum of its local computing time ($T_t^{\text{local}_k}$), and its transmission time ($T_t^{\text{transmission}_k}$), as in Equation 2.

$$l_t^k = T_t^{\text{local}_k} + T_t^{\text{transmission}_k} \tag{2}$$

1) Local computing time: It corresponds to the computation time required for a local iteration of the gradient descent method. We estimate it using client-specific resources: CPU frequency ($f_{t,k}$), number of cores (c_k), CPU cycles required to train one bit of data (g_k), and the size of the local data in bits held ($|D_{t,k}|$) during training round t, as shown in Equation 3.

$$T_t^{\text{local}_k} = \frac{|D_{t,k}| g_k}{c_k f_{t,k}} \tag{3}$$

2) Transmission time: It represents the duration needed to transfer the local model parameters from the client to the server. We estimate it using the bandwidth ($b_{t,k}$) of client k at training round t, and the size of its local model ($\sigma(m_k)$), as defined in Equation 4.

$$T_t^{\text{transmission}_k} = \frac{\sigma(m_k)}{b_{t,k}} \tag{4}$$

B. Handling Static Heterogeneity: Reputation and Utility

We introduce a reputation-based utility function as the objective function to handle static heterogeneity.

1) Utility function: The utility function, denoted as ζ, serves as a measure of a client's local update relevance to the global model. It's based on normalized model divergence [4] defined in Equation 5.

$$d(w_t^k, w_t) = \frac{1}{|w_t|} \sum_{j=1}^{|w_t|} \left| \frac{w_t^{k,j} - w_t^j}{w_t^j} \right| \tag{5}$$

where w_t represents the weights of a model during training round t, and $w_t^{k,j}$ and w_t^j is the j-th model weight of client k and the global model at training round t, respectively.

We therefore define our utility function in Equation 6. P represents the performance measure used in our study, such as accuracy and F1 score.

$$\zeta_t^k = \begin{cases} e^{-|d(w_t^k, w_t)|}, & \text{if } P_t > P_{t-1} \\ 1 - e^{-|d(w_t^k, w_t)|}, & \text{if } P_t \leq P_{t-1} \end{cases} \tag{6}$$

Our utility function is defined in two cases. The first case corresponds to an improvement in the global model performance, where we give a high score to clients whose local model parameters closely match the global model. We achieve this using the function $f(x) = e^{-|x|}$, where $x = d(w_t^k, w_t)$, which assigns a score close to 1 when x is near 0. The second case represents a deterioration in the global model performance, where we assign a low score to clients whose local model parameters closely match the global model. We achieve this using the function $g(x) = 1 - e^{-|x|}$, where $x = d(w_t^k, w_t)$.

2) Reputation-based utility function: Recently, researchers have extensively explored the adoption of reputation in client selection systems within the realm of FL [15]. We propose a hybrid approach, leveraging the advantages of both reputation and utility functions, defined in Equation 7.

$$\Psi_t^k = \lambda \zeta_t^k + (1 - \lambda) \Psi_{t-1}^k \tag{7}$$

We address both static and system heterogeneity by reformulating the reputation-based utility function in Equation 8, where α_1 and α_2 respectively denote the importance assigned to performance maximization and latency minimization.

$$\Psi_t^k = \lambda(\alpha_1 \zeta_t^k - \alpha_2 l_t^k) + (1 - \lambda) \Psi_{t-1}^k \tag{8}$$

The inclusion of a negative sign in front of latency serves to guide our RL agent to its minimization.

C. DDQL based client selection

1) Markov decision process formulation (MDP): To apply RL to client selection, it's crucial to formalize it as an MDP.

1) **State**: We introduce a vector representation of the state at each training round t, described in Equation 9.

$$s_t = \left(s_t^1, s_t^2, \ldots, s_t^N\right) \tag{9}$$

Where s_k represents the state of client k at training round t and is defined in Equation 10.

$$s_t^k = \left(w_t^k, n_t^k, c_k, f_{t,k}, b_{t,k}\right) \tag{10}$$

In which $w_t^k, n_t^k, c_k, f_{t,k}, b_{t,k}$ represent respectively the model weights, data amount, number of processor cores, CPU frequency and the bandwidth of client k at training round t. However, using the weight vector directly is impractical due to its large size. Thus, we apply PCA for dimensionality reduction of the weight vector [8].

2) **Action**: The action is a vector of N booleans, each denoting the selection status of a specific client, as defined in Equation 11.

$$a_t = \{i\} \times N, \text{ where } i \in \{0, 1\} \tag{11}$$

Selecting a subset of size U each round with the above representation yields C_N^U combinations, causing computational issues. We address this using multi-action RL, detailed in Section III-C2.

3) **Reward**: The reward is a vector of length U indicating the number of clients selected in each training round.

$$r_t = \left(r_t^1, r_t^2, \ldots, r_t^U\right) \tag{12}$$

Each element in this vector defines the reward assigned to client k during training round t, equal to Ψ_t^k.

2) Double Deep Q-learning Adaptation: we utilize the DDQL algorithm [16], comprising two neural networks: the main and target networks. The main network is used for training, while the target network evaluates actions for the next state and is periodically updated every P steps. Our DDQL agent incorporates a replay memory mechanism to mitigate correlations between consecutive experiences.

The RL learning problem can be framed as minimizing the Mean Squared Error (MSE) between the target and the approximated values, which is expressed as in Equation 13.

$$\text{MSE= } L_t^k\left(\boldsymbol{\theta}_t\right) = \left(Y_t^k - Q\left(\boldsymbol{s}_t, a_k; \boldsymbol{\theta}_t\right)\right)^2 \tag{13}$$

where Y_t^k is the target value at round t for action a_k determined by the Double Q-learning equation. Figure 1 depicts the training process of the DDQL networks.

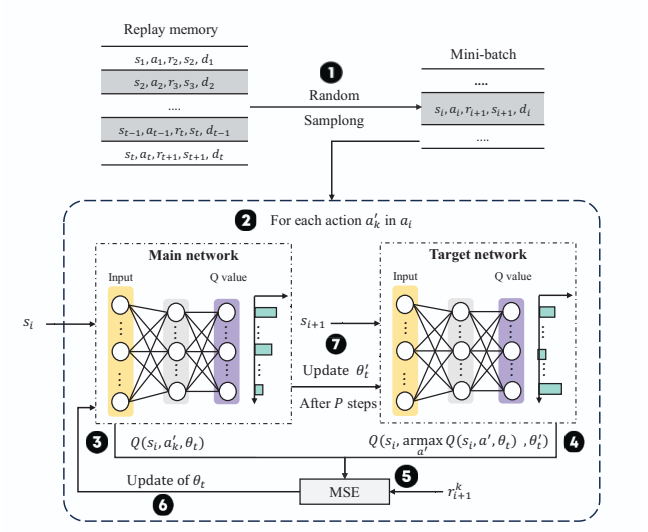

Fig. 1. DDQL training: 1- Random sampling from replay memory, 2- Iterating every sub-action a'_k of the global action a_t, 3, 4- Determining Q values, 5- MSE calculation, 6,7- Main/Target network update.

IV. Evaluation Methodology and Results

In this section, we present the evaluation of our proposed approach using three benchmark datasets: MNIST, CIFAR-10, and MobiACT for Fall Detection. We compare our approach with existing methods, namely FedAVG [5], and FAVOR [8] to assess its effectiveness and performance.

A. Experimental Methodology

For MNIST and CIFAR-10, we train a CNN with two 5x5 convolution layers followed by 2x2 max pooling. While for MobiAct. a two-layer GRU with 256 hidden units is employed.

To address static heterogeneity in FL, we focused on non-iid data. We used the FedLab [17] framework to generate three divisions of MNIST and CIFAR-10 datasets. For MobiAct, we proposed three divisions: *Volunteer-based division* in which each volunteer is treated as an individual client residing in their respective smart home, *Label-skew-8* and *Label-skew-4* where clients are assigned 8 and 4 activities, respectively. For system heterogeneity, we introduced variations in the edge equipment and transfer protocols to simulate various platforms, including Intel Core processors, TPUs, Raspberry Pi, Jetson Nano, Jetson TX2, Jetson Xavier NX, and FPGA Xilinx ZCU102.

B. Overall Results

Table I showcases multiple tests conducted on the non-iid divisions of CIFAR-10 and MNIST datasets. The evaluation metrics employed include accuracy(%) and latency(s).

In the CIFAR-10 dataset, specifically the Hetero Dirichlet 0.8 division, FLASH-RL outperformed FedAVG and FAVOR in accuracy, with improvements of 2.26% and 2.67%, respectively, while reducing latency by 0.78% and 2.03%. In Hetero Dirichlet 0.5 division, FLASH-RL improved accuracy by 1.26% compared to FedAVG and significantly reduced latency by 24.83% and 24.67% compared to FedAVG and FAVOR. In the Shards division, FLASH-RL improved accuracy by 3.36% over FedAVG and 0.86% over FAVOR, with latency reductions of 10.76% and 11.23%, respectively. For MNIST, while FAVOR achieved similar accuracy to FLASH-RL, our approach significantly reduced latency, such as a 11.20% reduction in Hetero Dirichlet 0.8 compared to FedAVG and 12.41% reduction compared to FAVOR. These results highlight the effectiveness of our method in striking a desirable balance between maximizing accuracy and minimizing latency.

Then we assess the convergence speed, which represents the number of rounds required to achieve specific target performance (52% for CIFAR-10 Hetero Dirichlet 0.8, 50.5% for Hetero Dirichlet 0.5, and 47% for Shards partition), as presented in Figure 2. We can observe that in Hetero Dirichlet 0.8 division, FLASH-RL significantly reduced training rounds by 60.44% compared to FedAVG and a remarkable 76% compared to FAVOR. In Hetero Dirichlet 0.5 division, we reduced rounds by 28.97% compared to FedAVG, while achieving latency reduction. In Shards division, FLASH-RL cut training rounds by 49.28% over FedAVG and 26.70% over FAVOR.

TABLE I
OVERALL RESULTS ON SEVERAL DATASET DIVISIONS COMPARED TO FEDAVG [5] AND FAVOR [8].

Dataset	Division	#Training rounds	FeAvg [5]		FAVOR [8]		FLASH-RL	
			Accuracy	Latency	Accuracy	Latency	Accuracy	Latency
CIFAR-10	Hetero Dirchlet 0.8	300	52.24	383.43	51.83	388.36	**54.50**	**380.45**
	Hetero Dirchlet 0.5	300	50.99	383.92	**52.90**	382.85	52.25	**288.59**
	Shards	300	47.84	378.61	50.34	380.59	**51.20**	**337.86**
MNIST	Hetero Dirchlet 0.8	100	99.04	676.89	**99.09**	682.22	99.06	**601.07**
	Hetero Dirchlet 0.5	100	98.99	674.13	**99.04**	672.77	98.90	**621.06**
	Non-iid label	150	98.89	1002.57	**99.02**	1015.77	98.74	**979.79**

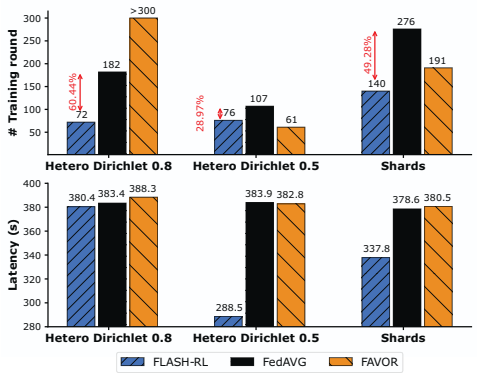

Fig. 2. Comparison of training rounds and end-to-end latencies on CIFAR-10.

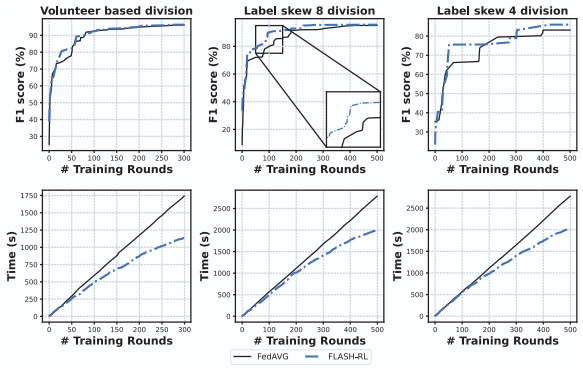

Fig. 3. General evolution of FL processes on MobiAct.

C. Case study

Figure 3 showcases the training results obtained on MobiAct divisions. The evaluation metrics employed include F1-score(%) and latency(s). The figure shows FLASH-RL's faster convergence to F1-score targets compared to FedAVG. For instance, in label-skew-8 division, FedAVG takes 417 rounds to reach a 95% F1-score target, while FLASH-RL achieves it in 228 rounds, a 45.32% reduction. In the label-skew-4, FedAVG needs 400 rounds for an 83% F1-score target, but FLASH-RL achieves it in just 86 rounds, a substantial 78.5% reduction. Additionally, FedAVG exhibits linear latency, while FLASH-RL maintains a constant latency, highlighting its effectiveness in balancing F1-score and latency.

V. CONCLUSION

In this study, we introduced FLASH-RL, a novel FL framework employing DDQL to tackle system and static heterogeneity. It selects clients based on their performance and latency using a reputation-based utility function. We also enhanced DDQL to expedite convergence. FLASH-RL outperforms existing methods, significantly reducing latency (up to 24.82%) and training rounds (up to 60.44% compared to FedAVG and 76% compared to FAVOR).

REFERENCES

[1] S. EK, F. PORTET, P. LALANDA, and G. VEGA, "A federated learning aggregation algorithm for pervasive computing: Evaluation and comparison," in *PerCom*, 2021, pp. 1–10.

[2] S. Kwatra and V. Torra, "A k-anonymised federated learning framework with decision trees," in *ESORICS*, vol. 13140, 2021, pp. 106–120.

[3] Y. Qin and M. Kondo, "Mlmg: Multi-local and multi-global model aggregation for federated learning," in *PerCom Workshops*, 2021, pp. 565–571.

[4] L. Fu, H. Zhang, G. Gao, H. Wang, M. Zhang, and X. Liu, "Client selection in federated learning: Principles, challenges, and opportunities," 2022.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *AISTATS*, 2017, pp. 1273–1282.

[6] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," vol. 37, 2020, pp. 50–60.

[7] 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.

[8] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing Federated Learning on Non-IID Data with Reinforcement Learning," in *IEEE Conference on Computer Communications*, 2020, pp. 1698–1707.

[9] H. Zhang, Z. Xie, R. Zarei, T. Wu, and K. Chen, "Adaptive Client Selection in Resource Constrained Federated Learning Systems: A Deep Reinforcement Learning Approach," vol. 9, 2021, pp. 98 423–98 432.

[10] G. Rjoub, O. A. Wahab, J. Bentahar, R. Cohen, and A. S. Bataineh, "Trust-Augmented Deep Reinforcement Learning for Federated Learning Client Selection," *A Journal of Research and Innovation*, pp. 1–18, 2022.

[11] S. Q. Zhang, J. Lin, and Q. Zhang, "A Multi-Agent Reinforcement Learning Approach for Efficient Client Selection in Federated Learning," in *AAAI*, vol. 36, 2022, pp. 9091–9099.

[12] I. W. W. Wisesa and G. Mahardika, "Fall detection algorithm based on accelerometer and gyroscope sensor data using recurrent neural networks," vol. 258, 2019, p. 012035.

[13] D. C. Yacchirema, J. S. de Puga, C. E. Palau, and M. Esteve, "Fall detection system for elderly people using iot and ensemble machine learning algorithm," vol. 23, no. 5-6, 2019, pp. 801–817.

[14] Q. Wu, X. Chen, Z. Zhou, and J. Zhang, "Fedhome: Cloud-edge based personalized federated learning for in-home health monitoring," vol. 21, no. 8, 2022, pp. 2818–2832.

[15] Y. Wang and B. Kantarci, "A novel reputation-aware client selection scheme for federated learning within mobile environments," in *CAMAD*, 2020, pp. 1–6.

[16] S. Latif, H. Cuayáhuitl, F. Pervez, F. Shamshad, H. S. Ali, and E. Cambria, "A survey on deep reinforcement learning for audio-based applications," *Artificial Intelligence Review*, Jul. 2022.

[17] D. Zeng, S. Liang, X. Hu, H. Wang, and Z. Xu, "Fedlab: A flexible federated learning framework," *Journal of Machine Learning Research*, vol. 24, no. 100, pp. 1–7, 2023.