FedGP: Genetic Programming for Evolutionary Aggregation in Federated Learning with non-IID data

Elia Pacioni[1,2][0000−0002−1557−4870], Francisco Fernández De Vega[1][0000−0002−1086−1483], and Davide Calvaresi[2][0000−0001−9816−7439]

[1] Universidad de Extremadura, Av. Santa Teresa de Jornet, 38. 06800 Mérida, Spain
eliapacioni@unex.es, fcofdez@unex.es
https://www.unex.es

[2] University of Applied Sciences and Arts of Western Switzerland (HES-SO Valais/Wallis), Rue de l'Industrie 23, 1950 Sion, Switzerland
elia.pacioni@hevs.ch, davide.calvaresi@hevs.ch
https://www.hevs.ch

Abstract. Federated Learning (FL) represents a distributed, privacy-preserving machine learning (ML) paradigm that enables decentralized model training across multiple clients. While traditional aggregation techniques, such as Federated Averaging (FedAVG), have demonstrated effectiveness, they often struggle in Not Independent and Identically Distributed (non-IID) scenarios, where data distributions vary significantly among clients. To address these limitations, this study introduces FedGP, a novel aggregation strategy based on Genetic Programming (GP). FedGP dynamically evolves aggregation functions, enabling adaptive and personalized model updates that better capture the heterogeneity inherent in distributed data. The proposed method is evaluated on the PathMNIST dataset, employing a comprehensive experimental design comprising 24 configurations, including 8 setups with FedAVG and 16 with FedGP. The comparative analysis highlights FedGP's superior generalization capabilities and reduced biases, outperforming FedAVG in terms of accuracy. These results position FedGP as a robust and scalable solution for real-world FL applications, particularly in environments characterized by data heterogeneity.

Keywords: Federated Learning · Genetic Programming · Model Aggregation · FedGP · FedAVG

1 Introduction

FL was introduced by Google in 2016 as a collaborative ML paradigm that trains models across decentralized data sources while keeping data localized on client devices [1]. Designed to address growing privacy and confidentiality concerns, FL enables the development of large-scale models without requiring data transfer to a central server [2]. A notable application is Google's Gboard, which uses

data from Android devices to train predictive text models locally, enhancing user experience while maintaining privacy [3]. Beyond mobile applications, FL has seen significant adoption in sectors such as healthcare [4] and finance [5, 6], where sensitive data demands robust privacy measures. At the core of FL is the aggregation of models, a pivotal step in the distributed training process. During aggregation, client devices send locally trained models (or their weights) to a central server, which synthesizes them into a global model using a predefined aggregation strategy. This process maintains data locality, reducing communication overhead and ensuring client privacy. In 2017, McMahan et al. [1] proposed the de-facto standard approach, namely Federated Averaging (FedAVG), that combines client updates through weighted averaging based on local data sizes. However, FedAVG faces significant challenges in non-IID settings, where data distributions vary across clients [7, 8]. In such cases, it struggles to effectively capture local data peculiarities, potentially resulting in a global model with reduced generalization capability [9, 10]. Simplified variants of FedAVG, which eliminate client weighting, have been explored to address privacy concerns, but they often fail to address data heterogeneity effectively. Improving model aggregation in FL remains a critical challenge. Recent studies highlight the need for techniques that accommodate client-specific variability and adapt to the dynamic nature of distributed systems [9, 10]. Enhancing aggregation mechanisms is essential for boosting global model accuracy and robustness, particularly in complex, heterogeneous environments.

This paper introduces FedGP (Federated Genetic Programming aggregation), a novel approach that leverages GP to address the limitations of traditional aggregation methods [11]. GP, a well-known ML evolutionary algorithm, is well-suited for adaptive and complex problems. During the last decades, it has shown flexibility to adapt to many different tasks, and has demonstrated its capabilities for addressing symbolic regression problems. This particular kind of problem is the inspiration for the problem we address: the dynamic evolution of aggregation strategies in the context of FL. As we show below, FedGP reduces biases and enhances generalization, surpassing the constraints of averaging-based methods.

The remainder of this paper is organized as follows: Section 2 outlines the theoretical background and technologies relevant to FL and GP. Section 3 details the methodology, including the FedGP framework, its evolution process, and selection criteria. Experimental results on the PathMNIST dataset [12], focusing on non-IID scenarios, are discussed in Section 4, comparing FedGP with FedAVG. Finally, Section 5 summarizes the findings, highlights contributions, and proposes avenues for future research.

2 State of Art

Since its introduction, FL has enabled decentralized ML across mobile and IoT (Internet of Things) systems, utilizing the vast amounts of locally generated data without requiring centralization. This paradigm has revolutionized distributed learning, with applications in mobile systems (e.g., GBoard for typing predic-

tion [3], Siri for text prediction and speech recognition [13]), healthcare (e.g., clinical data aggregation [14]), finance, and beyond. FL operates through an iterative process where client devices train local models on private data. Instead of sharing raw data, only model updates (weights or gradients) are sent to a central server for aggregation, preserving privacy [2]. FL can follow either a centralized or decentralized architecture [15]. Centralized FL relies on a server to aggregate local patterns, while decentralized FL distributes this task among devices, enhancing robustness against server failures or attacks. FL can also utilize horizontal or vertical data partitioning [16, 17]: in horizontal partitioning, clients hold data with similar attributes but different examples, while in vertical partitioning, clients have different attributes but the same examples, requiring complex integration techniques. Model updates in FL occur synchronously or asynchronously [17]. Synchronous FL waits for all clients to finish training before aggregation, which can cause delays in heterogeneous networks. Asynchronous FL updates models as client contributions arrive, improving efficiency but introducing challenges in stability. Security and privacy in FL are bolstered by differential privacy, secure multiparty computation (SMC), and homomorphic encryption, which protect user data during model updates [18, 19]. Techniques like gradient compression and quantization further optimize communication in low-bandwidth environments. Aggregation is at the core of FL. Federated Averaging (FedAVG) is the most widely used method, combining local updates through a weighted average based on client data sizes [20]. However, FedAVG struggles in non-IID scenarios where data distributions vary across clients, leading to suboptimal global models. Alternative approaches, such as local customization and meta-learning, have been explored to address these limitations [21]. Another method, FedSGD, updates models using stochastic gradients instead of averaging but is less commonly adopted [22]. Model adaptability remains a critical challenge in FL, particularly in non-IID environments and resource-limited settings. Meta-learning has shown promise in enabling global models to quickly adapt to new contexts with minimal data [21]. Additionally, bio-inspired algorithms have been explored to optimize transmission efficiency and model performance [23].

2.1 Genetic Programming and its role in FL

Introduced by Koza et al. [24], GP excels in optimizing non-linear functions and dynamically generating solutions. Since its inception, various versions of GP have been introduced, each contributing unique methodologies and insights to the field. These include (i) Linear GP [25], which structures programs as linear sequences of instructions, thereby enhancing their interpretability and optimizing performance; (ii) PushGP [26], a variant that employs the Push programming language, specifically designed to handle complex data structures and streamline the manipulation of evolving programs; (iii) Cartesian GP [27], which represents programs as acyclic directed graphs, affording greater flexibility in structural representation and facilitating the modeling of complex functions and systems; (iv) Geometric Semantic GP [28], which incorporates geometric operators to improve solution-finding efficiency and yield more robust outcomes than conven-

tional methodologies; (v) Grammatical Evolution [29], which leverages formal grammars to direct program evolution, thus enabling the generation of solutions in specific programming languages through the encoding of grammatical rules; and (vi) Genetic Improvement [30], which utilizes evolutionary techniques to enhance existing software, thereby augmenting performance, energy efficiency, or adaptability to new platforms while preserving the software's original functionality.

GP has demonstrated success across diverse domains, including creative applications (e.g., designing a commemorative coin in Portugal [32]), music transcription [33], learning the behaviors and generate controllers in the gaming field [34], in healthcare to improve medical image analysis [35], and to predict stock option prices in finance [36].

Notably, GP addresses a variety of problems, with the symbolic regression [31] being particularly salient, as it seeks to develop mathematical models that accurately represent a given set of data points defining the model. As described before, in FL, we need a mathematical function -the aggregation function- to work with multiple data, which somehow resembles the symbolic regression approach frequently addressed by means of GP. GP can be thus leveraged to evolve aggregation strategies tailored to heterogeneous data and varying client resources.

Its ability to adapt to complex scenarios makes it well-suited for enhancing FL aggregation methods. Despite advances in aggregation, traditional approaches like FedAVG show clear limitations [2], particularly in non-IID settings [5, 6]. This paper proposes a novel use of GP to address these challenges, focusing on two key research questions: RQ1: How can GP improve model aggregation in FL, particularly in non-IID contexts? RQ2: What is the impact of parameter and hyperparameter configurations on the performance of FedGP compared to FedAVG (e.g., batch size, aggregation frequency)?

3 Methodology

To address challenges related to the aggregation limitations of traditional methods, such as FedAVG, in FL scenarios characterized by non-IID distributions, we developed FedGP, a GP method designed to improve model aggregation. This approach dynamically optimizes aggregation functions to accommodate data variability across clients. In addition, to evaluate the performance of FedGP compared to FedAVG, we explore the effect of different parameter configurations and hyperparameters, such as batch size and aggregation frequency, by analyzing their impact on the generalization ability of the overall model.

The methodology, illustrated in Figure 1, is structured into six main blocks in agreement with the project's design phases. In particular, selection of the dataset study (3.1); technology stack definition (3.2); aggregation methods selection (3.3); FL and GP's parameters setup (3.4); environments configuration and tests execution (3.5); and results aggregation and analysis (3.6).

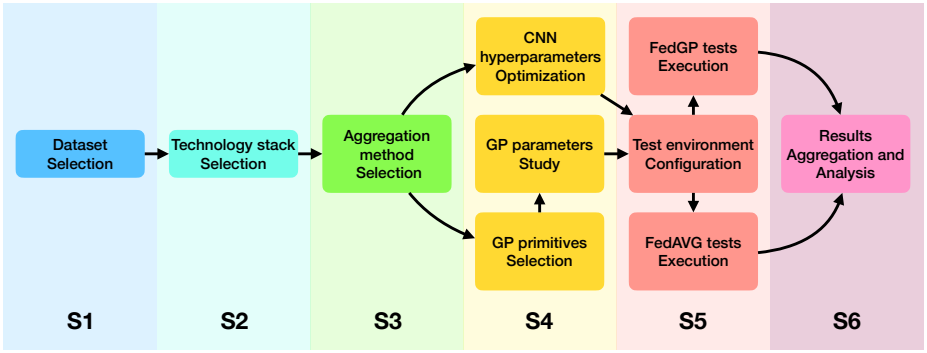

Fig. 1. Methodology organized in steps

3.1 S1: Dataset Selection

The first step concerns selecting a dataset suitable for FL experiments. Among FL's main application areas is the medical field, so the PathMNIST dataset from the MedMNIST collection [12], designed for medical image analysis, was chosen. Specifically, PathMNIST contains 107,180 colon histopathological images, each 28x28 pixels in size, it is important to note that in this study, we work directly with images and not with features pre-extracted in tabular data. The dataset is divided into nine classes representing different tissue types and histological structures. Derived from pathology slides, the dataset is intended for classification tasks in medical research, supporting automated diagnosis and analysis. The compactness and variety of classes make PathMNIST ideal for FL experiments. Figure 2 shows a sample montage of images from the dataset. The classes present are: (i) adipose, (ii) background, (iii) debris, (iv) lymphocytes', (v) mucus', (vi) smooth muscle, (vii) normal colon mucosa, (viii) cancer-associated stroma, (ix) colorectal adenocarcinoma epithelium.

3.2 S2: Technology stack selection

The entire project is developed using Python with PyTorch libraries to handle neural network (NN) training, operations on tensors, and DEAP [37] for GP. The model adopted for training the PathMNIST dataset is a convolutional neural network (CNN) suggested by the creators of the dataset. It has five convolutional layers followed by batch normalization, ReLU (rectified linear unit) activations, max-pooling, and a final section of fully connected layers. During the training process, accuracy and loss will be calculated for both training and validation data.

Individuals in FedGP are represented as expression trees, where each node corresponds to an operation or value. The tree structure is ideal for representing aggregation functions, as it allows mathematical and aggregation primitives to be dynamically combined to create complex functions. We describe below the functions and terminals used to build these expression trees.

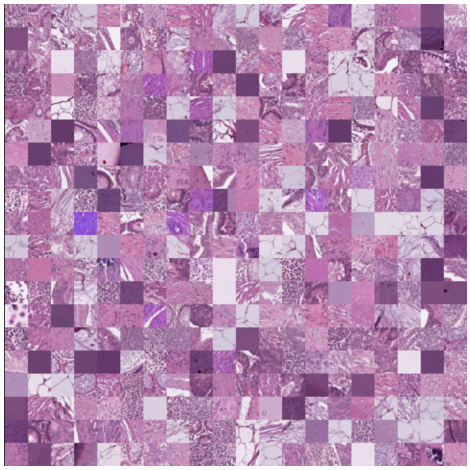

Fig. 2. PathMNIST image collage. Source: [12]

3.3 S3: Aggregation method Selection

Figure 3 shows the FL workflow. We have the central server and the user clients. Initially, the global model is trained on the server and sent to the clients. Once in the clients, local training begins on the user data. Using the synchronous, centralized version of FL, the server waits to receive the clients' models before aggregating them into a new global model.

In these expressions, one will always work with model weights. However, there are aggregation methods that work with gradients.

The aggregation step represents a crucial moment in the FL process.

As analyzed in the literature review, there are several methods. However, since this is the first step toward introducing a new aggregation method, unweighted FedAVG was chosen as the reference method for comparison. FedAVG is a logical choice since it represents the standard for aggregation in FL. The choice of the unweighted version is intended to provide privacy to users, so the server does not need to know how much data each client has processed and what type it is. FedAVG averages the weights of the clients and thus creates a new model to propagate. Overall, this study proposes the introduction of FedGP. The latter, unlike FedAVG, does not simply average but generates an expression tree that will then be applied to the clients to aggregate the weights and generate the new model to be propagated.

In the FL workflow, these operations are repeated cyclically unless a stop condition is established, such as a threshold on the metrics.

3.4 S4: FL and GP configurations

To ensure the proper functioning of the system, a prior study is carried out on the hyperparameters to be used for CNN. Specifically, we have:

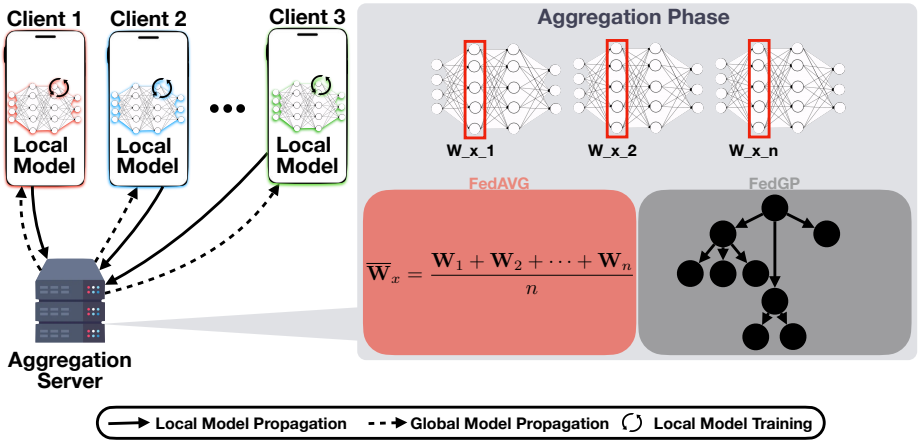

Fig. 3. FL workflow with explanation of FedAVG and FedGP aggregation methods.

- Learning rate: 0.0005. The optimizer's learning rate controls the weights' update rate during training.
- Momentum: 0.9. Momentum factor applied to improve convergence by reducing oscillations during optimization.
- An SGD optimizer is used to update CNN weights during training.
- Cost function: unweighted CrossEntropyLoss.

Next, the parameters for FL are set: (i) 5 clients; (ii) 2 epochs of global model training; (iii) 15 epochs of local training.

Before addressing the GP parameters, it is essential to define the primitives and the terminals that the GP can use during the evolutionary process. FedGP uses a specific set of primitives to work with PyTorch tensors. The functions `torch.sum, torch.sub, torch.mul`, and `torch.abs` are comprise in the PyTorch library, and `torch_protected_div, torch_mean, torch_median, torch_protected_sqrt` have been implemented for this study. For examples `torch_median` and `torch_mean` receive an arbitrary number of tensors, and place them in a stack. In turn, the averaging function is applied, so a new tensor with the same shape as the starting tensors and whose values are the mean and median of the tensors are obtained.

To handle data variability and prevent numerical errors, protected functions, such as protected division and protected square root, were introduced to avoid problematic situations such as division by zero and square roots of negative numbers.

$$F = \left\{ \begin{array}{l} \text{torch.sum, torch.sub, torch.mul,} \\ \text{torch\_protected\_div, torch\_mean,} \\ \text{torch\_median, torch.abs, torch\_protected\_sqrt} \end{array} \right\} \quad (1)$$

Equation 1 shows the available primitives, groupable into categories:

- Arithmetic operations: torch.add, torch.sub, torch.mul, torch_protected_div
- Aggregation functions: torch_mean, torch_median
- Transformations: torch.abs, torch_protected_sqrt

These primitives allow a wide range of aggregation configurations to be explored.

As terminals, placeholders are set up to represent the clients (equation 2) that have submitted their weights. Thus, when the tree is processed by the fitness function, the placeholder is replaced with the appropriate tensor.

$$T = \{\text{CLIENT}_1, \text{CLIENT}_2, \ldots, \text{CLIENT}_n \quad \} \tag{2}$$

A study is also carried out over GP parameters, choosing: (i) Generations: 10; (ii) Elitism: 1; (iii) Mutation rate: 40%; (iv) Crossover rate: 60%

To balance the complexity of the generated aggregation functions and their generalization, a minimum depth of 1 and a maximum depth of 5 were set for the trees. These depth limits were also chosen to avoid trees that might be computationally burdensome. Trees are combined through a one-point crossover, in which a subtree of one individual is swapped with another, maintaining the syntactic validity of the trees. Mutation is done by generating a subtree and replacing it with a subtree of the individual. The elitism of size 1 is adopted, preserving the individual with the best fitness in each generation to ensure that the best solutions are preserved.

The fitness function performs model validation on a test dataset. Model validation consists of making predictions on the GP aggregated model, and the fitness value is the accuracy value obtained as a result of the inference. The test dataset used for inference comprises a subset of PathMNIST images not used in the training phase, with an IID distribution to evaluate the entire model equally.

3.5 S5: Tests Performed

The FL process was implemented in a virtualized environment, leveraging threads to simulate distributed execution rather than employing physically distributed devices. This approach was adopted to avoid introducing additional variables that were not instrumental to the study. All tests were conducted on a server running Ubuntu 20.04.6, equipped with two Intel® Xeon® Silver 4310 processors, 512 GB of RAM, and four NVIDIA A100 PCIe GPUs (40 GB memory each), using CUDA version 12.4 and Python version 3.11.10. The GP algorithm was executed on the CPU, while the training and validation of FL models, as well as the evaluation of GP individuals, were performed on the GPU. To emulate the federated setting, model training executions were conducted in isolation on the same machine, with no physical distribution of devices. This methodology allowed for an accurate investigation of FL dynamics in a controlled environment. Each configuration was executed 30 times to ensure the statistical robustness and reliability of the results.

All experiments are performed with horizontal data partitioning and centralized and synchronous FL architecture. Given the nature of the proposed

methodology, the results can be easily generalized to asynchronous decentralized architecture.

Table 1 shows the variable parameters used in the experiments. Combining the available parameters between FedAVG and FedGP, taking into consideration that the number of individuals is only relevant for GP. The batch size represents the amount of data processed at each iteration in the NN. The Weights Sending Frequency determines every how many epochs model weights are sent from clients to servers, thus when the aggregate model is created. Then, the GP algorithm will run whenever the model needs to be aggregated. The *iid* parameter represents the data distribution; if the *iid* variable is false, data will have a non-IID distribution; otherwise, it will IID distribution. Figure 4 shows an example of IID and non-IID data distribution. In the non-IID case, it is not guaranteed that all clients have at least one example for each class; this condition promotes the divergence of local models from the global model trained on all classes. Independent of data distribution, a test dataset containing data from each class is used when the model is evaluated. For FedGP, the use of 20 or 50 individuals for the population is proposed. Finally, we find the chosen aggregation methods: FedGP and FedAVG.

Table 1. Possible values for each variable parameter in the experiment configurations.

Parameter	**Values**
Batch Size	64, 128
Weights Sending Frequency	3, 5
IID	TRUE, FALSE
Individuals (FedGP only)	20, 50
Aggregation Method	FedAVG, FedGP

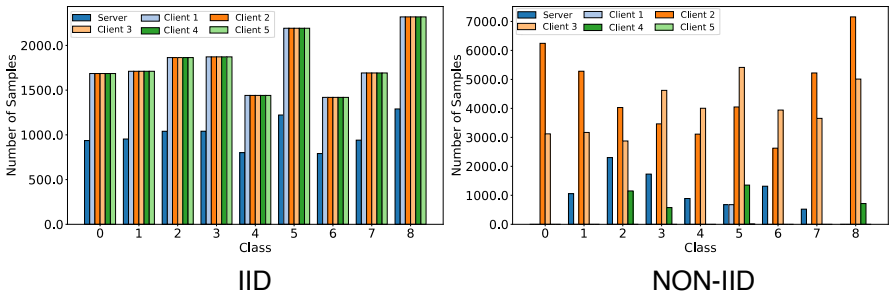

Fig. 4. Example of IID and non-IID data distribution for the PathMNIST dataset.

Table 2 summarizes the configurations used for the tests. Notably, the server is initialized with 10% of the dataset data to create the base model. The remain-

ing 90% is distributed among the clients. This design choice is intentional and not a fixed requirement; it allows the model to be heavily influenced by client contributions, enabling a clearer evaluation of aggregation effects. The experiments are divided into two groups: the first uses a batch size of 64, while the second uses a batch size of 128.

Table 2. Experiments' Configurations (Aggregation frequency is expressed in epochs).

ID	Batch Size	Agg. Freq.	IID	Ind.	Method	ID	Batch Size	Aggr. Freq.	IID	Ind.	Method
1	64	3	TRUE	-	FedAVG	13	128	3	TRUE	-	FedAVG
2	64	3	TRUE	20	FedGP	14	128	3	TRUE	20	FedGP
3	64	3	TRUE	50	FedGP	15	128	3	TRUE	50	FedGP
4	64	3	FALSE	-	FedAVG	16	128	3	FALSE	-	FedAVG
5	64	3	FALSE	20	FedGP	17	128	3	FALSE	20	FedGP
6	64	3	FALSE	50	FedGP	18	128	3	FALSE	50	FedGP
7	64	5	TRUE	-	FedAVG	19	128	5	TRUE	-	FedAVG
8	64	5	TRUE	20	FedGP	20	128	5	TRUE	20	FedGP
9	64	5	TRUE	50	FedGP	21	128	5	TRUE	50	FedGP
10	64	5	FALSE	-	FedAVG	22	128	5	FALSE	-	FedAVG
11	64	5	FALSE	20	FedGP	23	128	5	FALSE	20	FedGP
12	64	5	FALSE	50	FedGP	24	128	5	FALSE	50	FedGP

3.6 S6: Results Aggregation and Analysis

For each run, accuracy values for each epoch of the server and all clients are saved, along with data for each aggregation step. With the information obtained, mean and standard deviation are then calculated for server, client, and aggregate. So, the average among the 5 clients of all runs is averaged for each configuration. Afterward, the data is prepared for analysis and is eventually processed to generate a graph. Initially, a graph is created for each configuration, and subsequently, additional graphs are produced to directly compare two configurations in order to determine which one performs best.

4 Results and Analysis

The results of the experiments are shown in Figures 5 and 6 and represent the comparison between FedGP and FedAVG, in terms of accuracy, in the above configurations.

For FedGP to improve the compactness of the narrative, only the experiments involving 50 individuals are shown since they performed slightly better than the version with 20 individuals in each case.

4.1 IID Data Distribution

Figure 5 deals with the experiments with IID data distribution; experiments 1-3 and 7-9 have a batch size of 64, while experiments 13-15 and 19-21 have a batch size of 128. In this case, a larger batch size helps to decrease the results' variability by presenting a reduced standard deviation. Examining the comparison between FedGP (orange color) and FedAVG (blue color), we see that FedGP always obtains higher accuracy results than FedAVG. Results are always better regardless of the frequency of aggregation of the weights. In some cases, FedGP performs higher than the maximum value obtained by the clients; in each case, it obtains higher accuracy than the average of the clients.

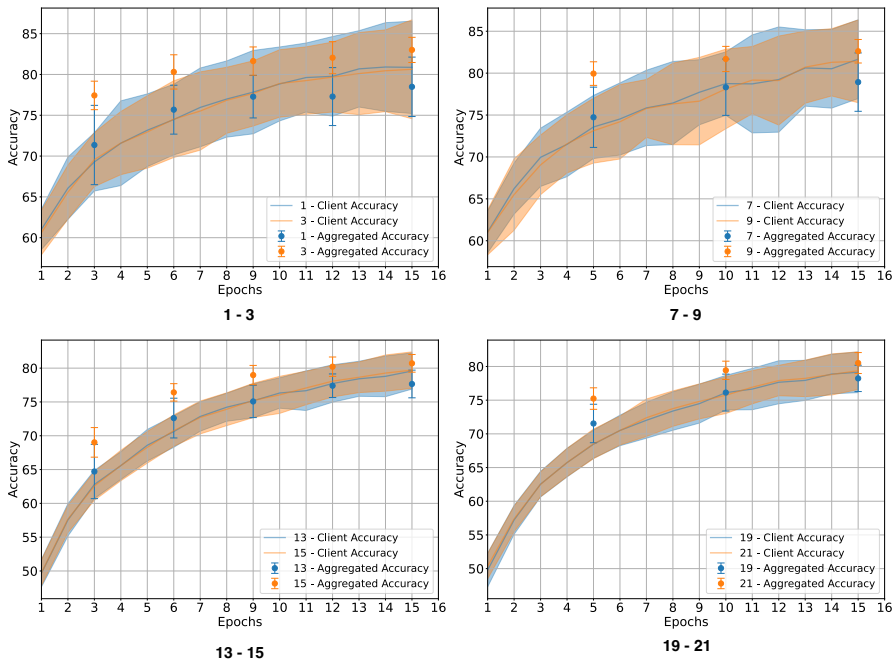

Fig. 5. Experiments with distribution of IID data. The blue represents the FedAVG, while the orange represents the FedGP.

4.2 non-IID Data Distribution

Focusing on the analysis of the experiments with non-IID data distribution, Figure 6, we notice that the standard deviation takes significant values; this is because clients are not guaranteed to have examples of every class in the dataset, so at the time of evaluation, clients are obtained that perform very well and

others of poor quality. This scenario is critical because it more closely represents the reality of FL application. Also, it is evident how FedGP always gets superior results than FedAVG. Moreover, with non-IID data, FedGP aggregates models with accuracy higher than the best of clients; this highlights the robustness and quality of the proposed aggregation method.

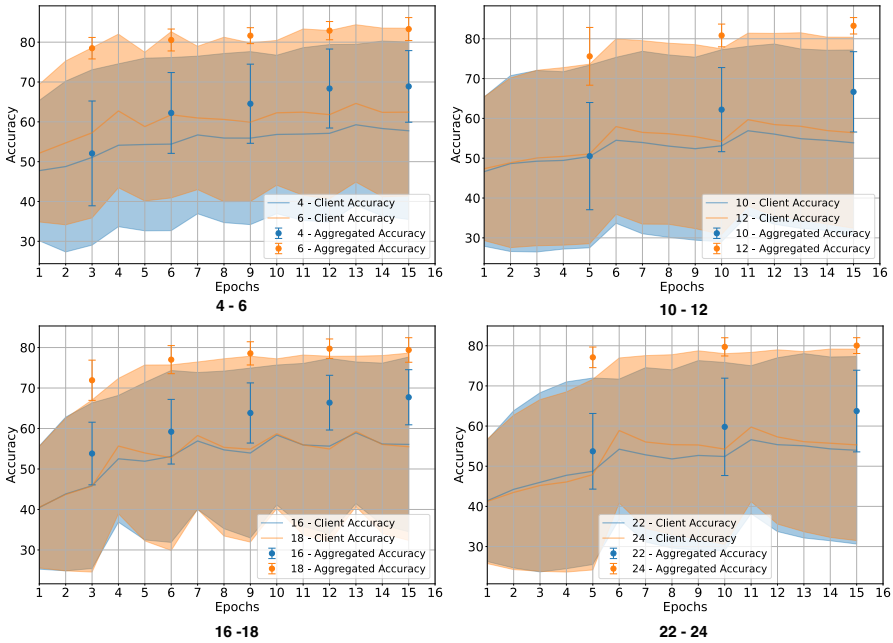

Fig. 6. Experiments with distribution of non-IID data. The blue represents the FedAVG, while the orange represents the FedGP. The numbers below each graph represent the configurations shown.

Thus, regardless of batch size, aggregation frequency, and data distribution, FedGP represents a promising aggregation method in FL.

Figure 7 represents an expression tree produced by GP. It shows how it combined the primitives of mean and median and selected the clients by effectively discarding client3. Therefore, it is interesting to study how, in addition to aggregating the weights, GP can select the clients that bring quality to the final solution while directly ignoring the others.

However, occasionally, GP produces higher quality but also bigger individuals than the one presented in Figure 7. For instance, Figure 8 shows the largest individual produced by GP, respecting the algorithm's depth limits. In the future, we plan to analyze the way GP produces the aggregation of clients which may be of help to design new strategies.

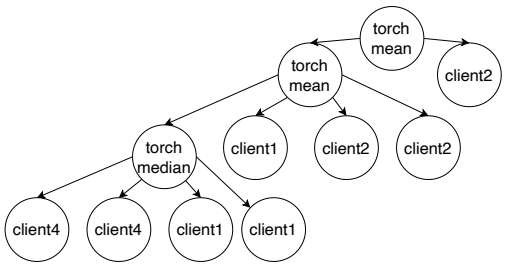

Fig. 7. Representation of an individual produced by the FedGP at the end of the evolutionary process.

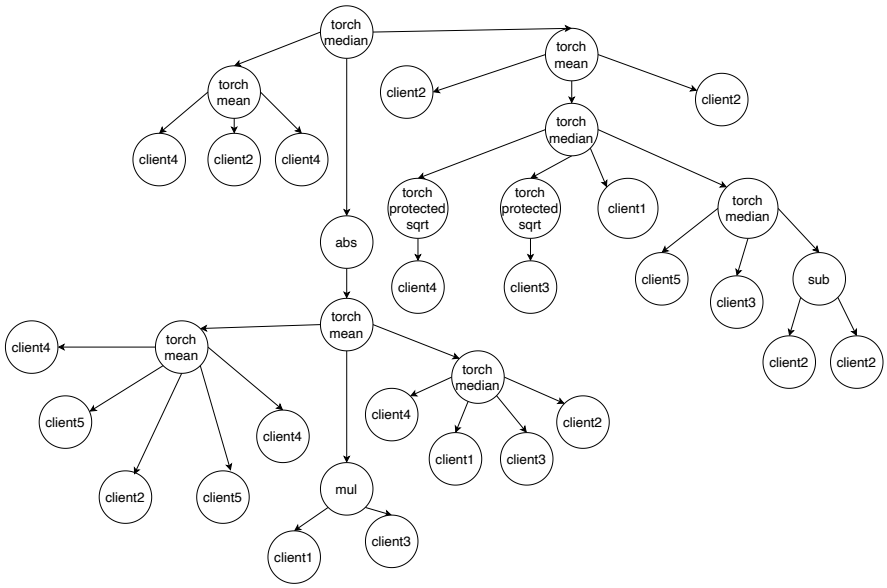

Fig. 8. Representation of the biggest individual produced by the FedGP.

Although only one network type was used for this experiment, FedGP is designed to be model-agnostic, thus applicable to any network architecture.

5 Conclusions

This paper introduced FedGP, a novel GP-based aggregation method for FL. To evaluate its effectiveness, FedGP was compared against FedAVG, the most widely adopted aggregation method in FL. The results demonstrate that FedGP consistently outperforms FedAVG and, in some cases, even surpasses the best-performing individual client. These findings highlight the robustness and supe-

rior performance of GP-driven aggregation compared to traditional global models.

A key insight from the experiments is that FedGP is minimally impacted by changes in aggregation frequency. Aggregation intervals were varied between every three epochs and every five epochs, yet FedGP outperformed FedAVG in both configurations, showcasing its adaptability to diverse operational settings.

Further testing in both IID and non-IID data environments confirmed FedGP's efficacy in addressing challenges associated with heterogeneous data distributions. While the experiments focused on a CNN, the model-agnostic design of FedGP indicates its potential applicability to other types of NNs.

This study marks a significant step forward in advancing aggregation techniques for FL through GP. To enhance the proposed method's validation, future work includes (i) comparing FedGP with alternative aggregation methods beyond FedAVG, (ii) exploring a gradient-based version of FedGP in place of the current weight-based approach, (iii) investigations into FedGP's resilience in privacy-preserving scenarios by analyzing its behavior in the event of noise injection; (iv) The study of the impact of dynamic variation in client data on FedGP; (v) Finally, expand the range of GP primitives and incorporating random ephemeral constants could further refine its ability to handle tensor data transformations.

In summary, FedGP offers a compelling improvement over FedAVG, paving the way for the development of innovative aggregation operators through evolutionary algorithms. This work underscores promising opportunities for enhancing the future of Federated Learning.

6 Acknowledgements

This work was partially supported by the HES-SO RCSO ISNet HARRISON grant (WP2), the Spanish Ministry of Economy and Competitiveness (PID2020-115570GB-C21, PID2023-147409NB-C22), funded by MCIN/AEI/10.13039/5011 00011033, and the Junta de Extremadura (GR15068).

References

1. McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and Agüera y Arcas, B. (2017). *Communication-Efficient Learning of Deep Networks from Decentralized Data.* In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS).* http://arxiv.org/abs/1602.05629.
2. Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawit, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Eichner, H., El Rouayheb, S., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konecný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Qi, H., Ramage, D., Raskar, R., Raykova, M., Song, D.,

Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H., and Zhao, S. (2021). *Advances and Open Problems in Federated Learning*. Now Foundations and Trends

3. A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated Learning for Mobile Keyboard Prediction," *arXiv preprint*, 2019. Available at: https://arxiv.org/abs/1811.03604.
4. Rieke, N., Hancox, J., Li, W., Milletarì, F., Roth, H. R., Albarqouni, S., Bakas, S., Galtier, M. N., Landman, B. A., Maier-Hein, K., Ourselin, S., Sheller, M., Summers, R. M., Trask, A., Xu, D., Baust, M., and Cardoso, M. J. (2020). *The future of digital health with federated learning. npj Digital Medicine*, 3(1), art. no. 119. https://doi.org/10.1038/s41746-020-00323-1.
5. Y. Liu, Z. Ai, S. Sun, S. Zhang, Z. Liu, and H. Yu, "FedCoin: A Peer-to-Peer Payment System for Federated Learning," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12500, Springer, 2020, https://doi.org/10.1007/978-3-030-63076-8_9.
6. G. Long, Y. Tan, J. Jiang, and C. Zhang, "Federated Learning for Open Banking," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12500, Springer, 2020, pp. https://doi.org/10.1007/978-3-030-63076-8_17.
7. W. Nie, L. Yu, and Z. Jia, "Research on Aggregation Strategy of Federated Learning Parameters under Non-Independent and Identically Distributed Conditions," in *Proceedings of the 2022 4th International Conference on Applied Machine Learning (ICAML)*, Changsha, China, 2022, pp. 41–48. DOI: 10.1109/ICAML57167.2022.00016.
8. H. Reguieg, M.E. Hanjri, M.E. Kamili, and A. Kobbane, "A Comparative Evaluation of FedAvg and Per-FedAvg Algorithms for Dirichlet Distributed Heterogeneous Data," in *Proceedings of the 2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM)*, Istanbul, Turkiye, 2023, pp. 1–6. DOI: 10.1109/WINCOM59760.2023.10322899.
9. X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the Convergence of FedAvg on Non-IID Data," *arXiv preprint*, 2020. Available at: https://arxiv.org/abs/1907.02189.
10. Wang, J., Liu, Q., Liang, H., Joshi, G., and Poor, H. V. (2020). *Tackling the objective inconsistency problem in heterogeneous federated optimization.* In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20)* (art. no. 638, pp. 1–13). Curran Associates Inc., Red Hook, NY, USA.
11. Pacioni, E., Fernández De Vega, F., Calvaresi C., "Towards a Meaningful Communication and Model Aggregation in Federated Learning via Genetic Programming". ICAART 2024.
12. Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H., and Ni, B. (2023). *MedMNIST v2: A large-scale lightweight benchmark for 2D and 3D biomedical image classification. Scientific Data*, 10(1), 41. Nature Publishing Group UK London.
13. Granqvist, F., Seigel, M., van Dalen, R., Cahill, Á., Shum, S., and Paulik, M. (2020). *Improving on-device speaker verification using federated learning with privacy.* https://arxiv.org/abs/2008.02651.
14. du Terrail, J. O., Léopold, A., Joly, C., Beguier, C., Andreux, M., Maussion, C., Schmauch, B., Tramel, E. W., Bendjebbar, E., Zaslavskiy, M., Wainrib, G., Milder, M., Gervasoni, J., Guérin, J., Durand, T., Livartowski, A., Moutet, K., Gautier, C., Djafar, I., Moisson, A.-L., Marini, C., Galtier, M., Bataillon, G., and Heudel, P.-E.

(2021). *Collaborative Federated Learning behind Hospitals' Firewalls for Predicting Histological Response to Neoadjuvant Chemotherapy in Triple-Negative Breast Cancer. medRxiv*, DOI: https://doi.org/10.1101/2021.10.27.21264834.
15. Martínez Beltrán, E. T., Pérez, M. Q., Sánchez, P. M. S., Bernal, S. L., Bovet, G., Pérez, M. G., Pérez, G. M., and Celdrán, A. H. (2023). *Decentralized Federated Learning: Fundamentals, State of the Art, Frameworks, Trends, and Challenges. IEEE Communications Surveys & Tutorials*, 25(4), 2983–3013, Fourthquarter 2023. DOI: 10.1109/COMST.2023.3315746.
16. Yang, Q., Liu, Y., Chen, T., and Tong, Y. (2019). *Federated Machine Learning: Concept and Applications. ACM Transactions on Intelligent Systems and Technology (ACM Trans. Intell. Syst. Technol.)*, 10(2), Article 12, 19 pages. DOI: https://doi.org/10.1145/3298981.
17. Xu, C., Qu, Y., Xiang, Y., and Gao, L. (2023). *Asynchronous federated learning on heterogeneous devices: A survey. Computer Science Review*, 50, 100595. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2023.100595.
18. Mugunthan, V., Polychroniadou, A., Byrd, D., and Balch, T. H. (2019). *SMPAI: Secure Multi-Party Computation for Federated Learning.* In 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada. https://www.jpmorgan.com/content/dam/jpm/cib/complex/content/technology/ai-research-publications/pdf-9.pdf.
19. Madi, A., Stan, O., Mayoue, A., Grivet-Sébert, A., Gouy-Pailler, C., and Sirdey, R. (2021). *A Secure Federated Learning framework using Homomorphic Encryption and Verifiable Computing.* In *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, Hamilton, ON, Canada, pp. 1–8. https://doi.org/10.1109/RDAAPS48126.2021.9452005.
20. Qi, P., Chiaro, D., Guzzo, A., Ianni, M., Fortino, G., and Piccialli, F. (2024). *Model aggregation techniques in federated learning: A comprehensive survey. Future Generation Computer Systems*, 150, 272–293. https://doi.org/10.1016/j.future.2023.09.008.
21. Fallah, A., Mokhtari, A., and Ozdaglar, A. (2020). *Personalized Federated Learning: A Meta-Learning Approach.* https://arxiv.org/abs/2002.07948.
22. Yuan, H.; Ma, T. Federated accelerated stochastic gradient descent. Adv. Neural Inf. Process. Syst. 2020, 33, 5332–5344.
23. Souza, M. M. de, Holm, A., Biczyk, M., and de Castro, L. N. (2024). *A Systematic Literature Review on the Use of Federated Learning and Bioinspired Computing. Electronics*, 13(16), 3157. DOI: https://doi.org/10.3390/electronics13163157.
24. John R. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press. http://mitpress.mit.edu/books/genetic-programming
25. Brameier, M., and Banzhaf, W. (2007). *Linear Genetic Programming.* Springer, New York, NY. DOI: https://doi.org/10.1007/978-0-387-31030-5.
26. Spector, L. (2001). *Autoconstructive Evolution: Push, PushGP, and Pushpop.* In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO).*
27. Miller, J. F., and Thomson, P. (2000). *Cartesian genetic programming.* In *Lecture Notes in Computer Science*, Vol. 1802, pp. 121–132. https://doi.org/10.1007/978-3-540-46239-2_9.
28. Moraglio, A., Krawiec, K., and Johnson, C. G. (2012). *Geometric semantic genetic programming.* In *Lecture Notes in Computer Science*, Vol. 7491 (Part 1), pp. 21–31.
29. O'Neill, M., and Ryan, C. (2001). *Grammatical evolution. IEEE Transactions on Evolutionary Computation*, 5(4), 349–358. DOI: https://doi.org/10.1109/4235.942529.

30. Petke, J., Haraldsson, S. O., Harman, M., Langdon, W. B., White, D. R., and Woodward, J. R. (2018). *Genetic Improvement of Software: A Comprehensive Survey. IEEE Transactions on Evolutionary Computation*, 22(3), 415–432, June 2018. https://doi.org/10.1109/TEVC.2017.2693219.
31. Augusto, D. A., and Barbosa, H. J. C. (2000). *Symbolic regression via genetic programming.* In *Proceedings, Vol. 1, Sixth Brazilian Symposium on Neural Networks*, Rio de Janeiro, Brazil, pp. 173–178. https://doi.org/10.1109/SBRN.2000.889734.
32. Machado, P., Martins, T., Correia, J., Santo, L. E., Lourenço, N., Cunha, J., Rebelo, S., Martins, P., and Bicker, J. (2024). *Designing Coins with Evolutionary Computation. SIGEVOlution*, 17(2), Article 1, 9 pages. https://doi.org/10.1145/3695933.3695934.
33. Miragaia, R., Fernández, F., Reis, G., and Inácio, T. (2021). *Evolving a Multi-Classifier System for Multi-Pitch Estimation of Piano Music and Beyond: An Application of Cartesian Genetic Programming. Applied Sciences*, 11(7), 2902. https://doi.org/10.3390/app11072902.
34. Wilson, D. G., Luga, H., Cussat-Blanc, S., and Miller, J. F. (2018). *Evolving simple programs for playing Atari games.* In *GECCO 2018 - Proceedings of the 2018 Genetic and Evolutionary Computation Conference*, pp. 229–236. https://doi.org/10.1145/3205455.3205578.
35. Langdon, W. B., Modat, M., Petke, J., and Harman, M. (2014). *Improving 3D medical image registration CUDA software with genetic programming.* In *GECCO 2014 - Proceedings of the 2014 Genetic and Evolutionary Computation Conference*, pp. 951–958. DOI: https://doi.org/10.1145/2576768.2598244.
36. Hsu, C.-M. (2011). *A hybrid procedure for stock price prediction by integrating self-organizing map and genetic programming. Expert Systems with Applications*, 38(11), 14026–14036. https://doi.org/10.1016/j.eswa.2011.04.210.
37. Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). *DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research*, 13, 2171–2175.