



FedGP goes open source: reproducible evolutionary aggregation for federated learning with FASTER

Elia Pacioni^{1,2} · Valerio Crocetti^{1,3} · Francisco Fernández de Vega² · Davide Calvaresi¹

Received: 15 July 2026 / Revised: 7 October 2026 / Accepted: 8 October 2026
© The Author(s) 2026

Abstract

Genetic Programming (GP) has recently been applied to evolve aggregation functions in Federated Learning (FL), instead of fixing them in advance. FedGP represents aggregation rules as symbolic expressions that evolve during training and has shown competitive performance under non-IID data. Progress on GP-based aggregation, however, has been constrained by two practical barriers: limited reproducibility in FL experimentation and the absence of a public FedGP implementation. This paper presents FASTER, an open-source platform that provides the first public implementation of FedGP and integrates it with standard baselines (FedAvg, weighted FedAvg, FedProx, and FedNova). FASTER provides a browser- and CLI-based configuration-driven workflow for controlled FL experiments with built-in or custom datasets, IID and non-IID partitions, client scheduling, churn simulation, live metric tracking, and structured artifact export. Each run is stored with its configuration, logs, metrics, metadata, checkpoints, and, for FedGP, the individuals evolved across generations. In a demonstration on FashionMNIST with non-IID data and client churn, FedGP outperforms all fixed baselines with statistically significant gains; GPU batch evaluation reduces the previously reported roughly ten-fold wall-clock overhead to about $3.3\times$ relative to fixed-rule methods. By placing evolved and fixed aggregation rules under identical controlled conditions, FASTER turns FedGP into a reusable, testable, and extensible research object for the evolutionary computation community.

Keywords Genetic programming · Federated learning · FedGP · FASTER · Aggregation methods

Extended author information available on the last page of the article

1 Introduction

Federated Learning (FL) has become a central paradigm for training machine learning models over distributed data without centralizing raw samples [1, 2]. In a typical FL setting, multiple clients train local models on private data and periodically share model updates with a coordinating server, which aggregates them into a global model. This paradigm is particularly relevant in domains such as healthcare [3], finance [4], mobile computing [5], and edge intelligence, where data are naturally distributed and often subject to privacy-related [6] regulatory, or operational constraints. Experimental outcomes depend on many interacting design choices, including the number of clients, partitioning strategy, degree of heterogeneity, client imbalance, local training schedule, aggregation frequency, model architecture, random seed, and evaluation protocol. Even small changes in these factors can substantially affect convergence, stability, and final performance. As a result, many studies still rely on ad hoc pipelines, making it hard to reproduce results, isolate the effect of a single variable, or compare aggregation strategies under equivalent conditions.

The aggregation phase is one of the most sensitive components of the FL pipeline. Classical methods such as Federated Averaging (FedAvg [1]) remain widely used because they are simple, scalable, and computationally efficient. Fixed aggregation rules, however, may be insufficient when client data are statistically heterogeneous (non-IID) [7], participation is irregular, or updates are noisy or corrupted. These limitations have motivated proximal (FedProx [8, 9]), normalized (FedNova [10]), robust [11, 12], personalized [13], and adaptive methods [14]. In this context, Genetic Programming (GP)-based aggregation has emerged as an alternative that evolves aggregation functions instead of fixing them a priori [15]. FedGP, introduced in prior work, represents aggregation rules as symbolic expressions and evolves them to combine client model parameters under heterogeneous conditions [15]. Subsequent studies compared FedGP with standard methods and further examined its behavior in non-IID and adversarial scenarios [12, 16]. However, no public FedGP implementation was available, preventing independent evaluation, extension, and reuse.

To make the idea concrete, FedGP treats aggregation not as a fixed formula chosen in advance, but as a program to discover. At each communication round, the server collects model updates from participating clients and combines them into a global model. Standard methods use predetermined expressions, most commonly uniform or sample-weighted averaging. FedGP instead represents the combination rule as a symbolic expression that takes client updates as input and returns the aggregated model. Aggregation therefore becomes an optimization problem nested within the federated loop. Concretely, each candidate rule is an expression tree whose leaves are the current-round client updates and whose internal nodes are tensor operations that combine them. A population of such trees evolves for several generations within each round, is scored on a small server-side validation set, and yields the best expression for global-model aggregation before training proceeds to the next round. Because the search is guided by validation performance rather than a fixed heuristic, the rule can adapt to statistical heterogeneity and irregular participation, where fixed rules often struggle. Section 4 details the implementation in FASTER (Federated Aggregation, Simulation, TEsting, and Reporting).

Several open-source frameworks already support FL experimentation. Flower [17], FedML [18], and FedScale [19] provide mature environments that emphasize scalability, system realism, and standardized benchmarking, from single-machine simulations to very large client populations and real edge devices. These platforms have substantially advanced FL research. They do not, however, provide native support for evolutionary aggregation in general or FedGP in particular, and they do not enforce the artifact structure used here for controlled aggregation studies: a fully specified scenario, aggregation as the single controlled variable, and run preservation as a self-contained reproducible object. This creates a practical limitation for FedGP and related GP-based aggregation research, where evolved rules should be compared directly and fairly with standard methods under identical conditions.

FASTER fills this gap. It prioritizes reproducibility and controlled comparison over deployment scale. This letter makes three contributions: (i) to the best of our knowledge, the first public release of FedGP, integrated as a native aggregation method and extended beyond the original prototype, notably with vectorized GPU evaluation; (ii) a reproducible workflow that makes the aggregation rule the single controlled variable in FL experiments; and (iii) to the best of our knowledge, the first evaluation of FedGP under client churn, where it ranks first in all ten paired repetitions with statistically significant gains over all fixed baselines.

2 The FASTER platform

FASTER is an open-source platform designed to support the configuration, execution, monitoring, and reporting of FL experiments. The source code, including the FedGP implementation, is publicly released¹ under the MIT license. Its purpose is to reduce the engineering overhead of controlled FL studies while preserving the information needed for reproducibility. FASTER provides a modular environment in which alternative federated scenarios can be defined, executed, compared, and exported through a common workflow.

The platform was motivated by the need to make FL experimentation more consistent, traceable, and comparable. Aggregation research in particular requires strict control of the environment, since changes in partitioning, client allocation, aggregation frequency, local training schedules, or random seeds can materially affect outcomes. FASTER addresses this need through a browser-based workspace, a configuration-driven backend, a job-management subsystem, and artifact storage.

2.1 Design goals

FASTER is built around three design goals: reproducibility, comparability, and usability. To ensure reproducibility, each experiment is defined through structured configuration parameters, created through the web interface or loaded from scenario files. For comparability, all aggregation methods execute within the same FL training pipeline under the same controlled conditions. Finally, to maximize usability,

¹ <https://github.com/HISTLab-hevs/FASTER>

the browser interface provides a guided workflow for configuring, launching, monitoring, and exporting experiments, while a command-line path through the training worker remains available for debugging, automation, and reproducibility-oriented workflows.

2.2 Architecture and experiment workflow

FASTER separates the browser workspace, where experiments are configured and monitored, from the backend that validates and queues runs, and from the training worker that executes them. The worker runs each experiment from a generated configuration file, and all outputs are persisted in a database (experiment state) and on the filesystem (artifacts, described in Sect. 3.3). The FASTER platform ships as Docker images that pin the execution environment, so rebuilding on a different machine recreates the intended software stack and supports rerunning experiments under recorded conditions. Deployment instructions are provided in the repository.

A typical FASTER experiment follows a workflow (Figs. 1 and 2) comprising six phases: dataset selection or upload (Fig. 1A), scenario configuration (Fig. 1B), run submission (Fig. 1C), live monitoring (Fig. 1D), post-run analysis (Fig. 2A), and artifact export (Fig. 2B). During configuration, the user defines a federated scenario in either a simplified mode with guided defaults or an advanced mode that exposes detailed controls for FL, scheduling, churn, and aggregation, including FedGP-specific parameters when selected (Sect. 3.1).

Once submitted, a run is validated, queued, and executed by the training worker, which initializes seeds, prepares partitions, runs the federated loop, and computes validation or test metrics at each round. Metrics stream to the interface for live moni-

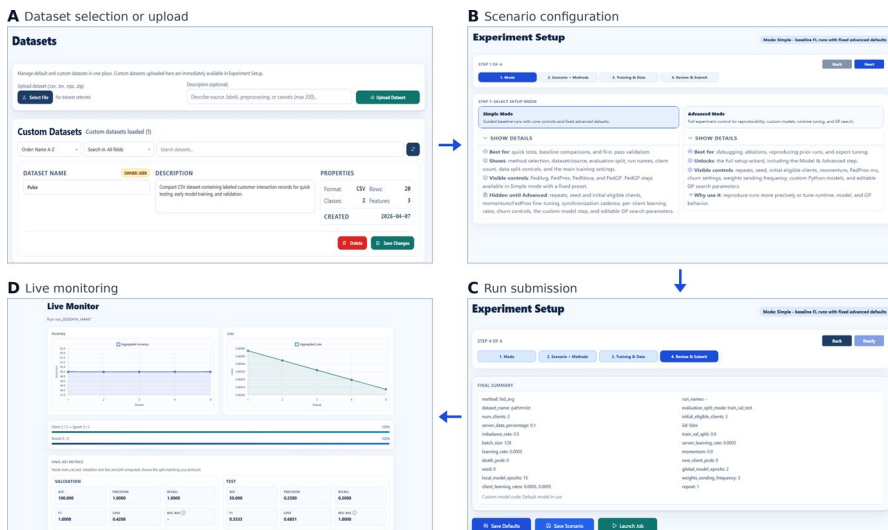


Fig. 1 FASTER user-facing workflow: configuration and execution. **A** Dataset selection or upload. **B** Scenario configuration: in advanced mode, users can configure more FL parameters and FedGP-specific settings. **C** Run submission. **D** Live monitoring: during execution, metrics are updated in real time

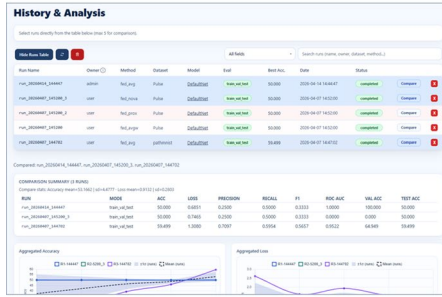
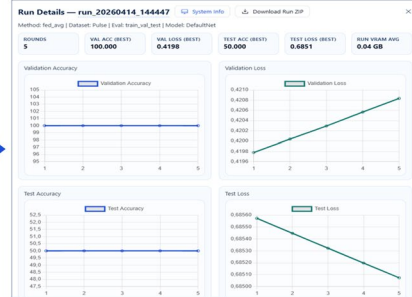
E Post-run analysis**F** Artifact export

Fig. 2 FASTER user-facing workflow: post-run analysis. **E** After completion, runs can be inspected, compared, or deleted. **F** Run details and artifact export

toring, helping users detect unstable or failed runs early, and are persisted as structured artifacts.

3 Reproducible FL experimentation

FASTER represents each FL experiment as a persistent, structured scenario that captures data, partitioning, participation, optimization, aggregation, evaluation, logging, and reporting, enabling reproducible runs and controlled comparisons across aggregation methods.

3.1 Scenario configuration

A FASTER scenario specifies the main components of an FL experiment: dataset, number of clients, model architecture, local training parameters, server-side warm-up, aggregation frequency, random seed, and selected aggregation method. Scenarios can be created through the web interface, passed through the API, or represented as structured configuration files. The training worker can also be executed from the command line. This configuration-driven design allows completed runs to be inspected, modified, relaunched, and shared.

The platform supports both built-in and custom datasets. Built-in datasets include MedMNIST [20] variants, FashionMNIST [21], CIFAR-10, and CIFAR-100 [22], while custom datasets can be provided in tabular or image-based formats. This enables evaluation of aggregation strategies on standardized benchmarks and user-provided data. FASTER supports both IID and non-IID partitions, with a configurable imbalance rate setting the severity of the non-IID case. Each class is split into equally sized shards, which are pooled and shuffled, so every shard holds a single class. Shards are then assigned to clients in proportion to weights drawn from a Dirichlet distribution with concentration parameter equal to the complement of the imbalance rate. The weights govern how much data each client holds, not which classes it sees. Label skew follows: a client with few single-class shards observes few classes. Beyond the

initial IID or non-IID partitioning, FASTER supports round-level control over how federated training data are consumed across rounds and clients.

A scenario also fixes client participation and the standard optimization and evaluation parameters. Users set the initial number of eligible clients and can simulate churn through dropout and new-client probabilities, applied independently to each client every round so that dropout is never permanent, letting the set of trainable clients change across rounds and aggregation be studied under non-stationary participation.

Local optimization, evaluation split, and the number of repeated runs are configured in the same way, with repetitions estimating variability due to seeds, partitioning, and initialization. The repository documents the full parameter list. Importantly, every parameter is stored with the run, so configurations can be re-executed from recorded settings instead of reconstructed by hand. In addition to the platform's default models, users can provide custom model code that defines the model-building function required by the worker.

3.2 Controlled aggregation comparison

FASTER executes all aggregation strategies through a single federated pipeline. Dataset split, client schedule, model architecture, optimizer settings, evaluation protocol, and output structure remain fixed across methods; therefore, the aggregation rule is the single controlled variable. The current aggregation layer covers both classical and adaptive methods. FedAvg provides the uniform averaging baseline, and weighted FedAvg (FedAvgW) accounts for the number of samples each client owns. FedProx adds a proximal term during local training to reduce client drift under heterogeneous data, while FedNova normalizes local updates when clients perform different amounts of local work. FedGP extends the layer with an evolved, rather than fixed, aggregation rule, and its method-specific parameters are part of the run configuration, like everything else (Sect. 4). The setup also lends itself to sensitivity analysis. A user can fix the scenario and vary only the aggregation method, or fix the method and vary the degree of non-IIDness, the churn probabilities, or the evolutionary configuration, and observe how each rule reacts to that single change.

The scenario can also be made dynamic. In automatic mode, federated training data are spread uniformly across rounds and clients. In manual mode, the user assigns a data percentage to each aggregation round and controls how that round's subset is distributed among clients. This supports simulations in which data availability changes over time, for example when clients join late with fresh data or when datasets grow as the federation progresses, and also cases where some clients dominate specific rounds while others contribute little or become inactive. Combined with churn, these controls reproduce realistic participation dynamics.

This round-level scheduling is distinct from GP generations. Aggregation rounds are part of the federated process and define when clients train locally and when the server aggregates. GP generations are part of the evolutionary process and define how candidate expressions evolve within a round. FASTER records both levels when FedGP is selected, enabling dynamic FL scenarios to be combined with an explicitly configured evolutionary search.

3.3 Metrics, artifacts, and reporting

During execution, FASTER records both live and persistent information for each run. Metrics are updated across federated rounds and can be inspected from the web interface while an experiment is running. Depending on the selected evaluation mode, validation and test metrics are stored for the global model together with client- and round-level information. FedGP uses validation metrics for fitness evaluation, while test metrics support post-hoc comparison and diagnosis of unstable configurations.

Each run produces a set of reproducibility artifacts. The configuration snapshot records the parameters used to launch the experiment. The metadata file stores information such as the software version, git hash, dependency versions, and seed. The execution log preserves the training trace. The metrics file stores per-client metrics, aggregated validation and test results, round-level information, and the live execution state. Model checkpoints record the global model across aggregation rounds, while dataset distribution files preserve information about the generated client partitions. For FedGP runs, additional files store evolved individuals across generations, making the evolutionary process inspectable after execution.

When repeated executions are performed, FASTER creates separate repetition directories and a summary file that aggregates resulting metrics. This supports more reliable comparisons across stochastic runs and enables reporting of average performance and variability. Completed runs can also be exported as compressed archives. This export mechanism supports sharing and independent inspection because the core elements required to understand and reproduce a run are preserved together. In this sense, FASTER treats a completed experiment not only as an interface result but as a self-contained, reproducible research object.

4 FedGP integration

This section details how FedGP is implemented in FASTER (Fig. 3). Beyond the original prototype [15], the implementation has been extended with vectorized GPU population evaluation, configurable fitness metrics, ephemeral constants, a larger primitive set, and multiple population-transfer strategies across rounds.

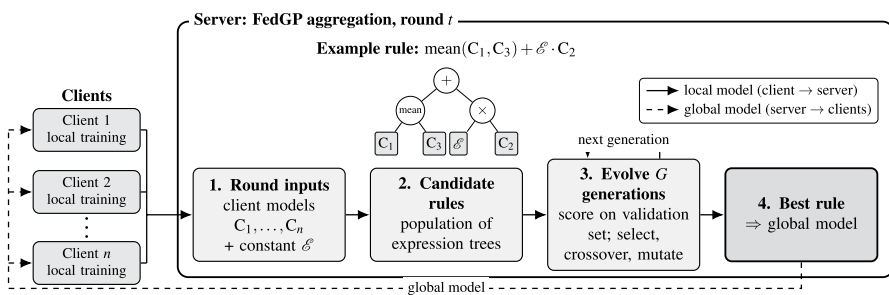


Fig. 3 FedGP in one aggregation round. Client models and an ephemeral constant form the inputs (1); a population of expression trees (2) is evolved for G generations, scored on the server-side validation set (3); the best tree yields the global model (4), broadcast to the clients

FedGP treats the aggregation rule as an evolvable program. Instead of applying a fixed formula, such as uniform or weighted averaging, the server evolves symbolic expressions that combine the client model parameters, thereby turning aggregation into a search problem embedded in the FL training loop. Evolution runs on the server, between the arrival of the client updates and the start of the next local-training phase: once the active clients have sent their weights, the terminal set is rebuilt from those updates, a population evolves for a configurable number of generations scored on the server-side validation set, and the best individual produces the global model for the next round. Clients take no part in the search and fitness never uses their data. Each candidate is an expression tree. The terminal set gathers the client model updates available in the round together with ephemeral constants:

$$T = \{\text{Client}_1, \dots, \text{Client}_n\} \cup \{\mathcal{E}\}, \quad (1)$$

where Client_i is the model update of the i -th participating client, n is the number of clients active in the round, and $\mathcal{E}$ denotes an ephemeral constant sampled from a configurable interval. The number of active clients, n , varies from round to round as participation changes under churn. The function set collects element-wise tensor operations:

$$F = \{+, -, \times, \div, |\cdot|, \sqrt{\cdot}, (\cdot)^{(\cdot)}, \log, \sin, \cos, \text{mean}, \text{median}\}, \quad (2)$$

where $\div$ and $\sqrt{\cdot}$ denote protected division and square root, $(\cdot)^{(\cdot)}$ is the power, and mean and median are computed across any subset of the client tensors. Each experiment enables the primitives it needs through the configuration, and the demonstration in Sect. 5 uses a subset of them. Tree depth is bounded at initialization and mutation grafts subtrees of limited height, which keeps bloat under control across generations.

The GP layer uses DEAP [23] for tree representation, genetic operators, and compilation of trees into callables. At the same time, each primitive is implemented as a PyTorch [24] operation, so individuals run directly on client state dicts without leaving the deep learning framework. On GPU, the full population is evaluated in a single vectorized forward pass. Aggregated weight sets of all valid individuals are stacked along a leading dimension and scored on the server-side validation data in one batch, removing the per-individual overhead that dominated earlier implementations and substantially reducing evolutionary runtime cost. Fitness is a configurable validation metric (accuracy, loss, macro precision, macro recall, or macro F1), whereas previous FedGP versions optimized only accuracy. Individuals whose programs fail, produce non-finite aggregated tensors, or generate aggregated parameters with a maximum absolute value above a configurable stability threshold are assigned the worst possible fitness instead of being evaluated. By default, this threshold is set to (10^6) ; the worst fitness is (0) for accuracy, precision, recall, and F1, and (-10^9) for the loss-based objective.

Between rounds, the population can be reinitialized from scratch or transferred, either in full or through elites, mutated elites, or a hybrid of elites, mutants, and random individuals. Under churn, the terminal set is rebuilt at each aggregation round to match the active clients, and transferred individuals referencing clients no lon-

ger present in the active set are treated as invalid, assigned the worst fitness rather than repaired or remapped, and removed by selection. Initialization schemes, genetic operators, selection, elitism, and early stopping are also configurable, and the full list of options is documented in the repository.

Every FedGP run stores individuals from each generation, along with fitness values, alongside standard artifacts, so evolved expressions can be inspected after execution and linked to round-level metrics. This inspectability supports future work on primitive design, transfer across rounds, and multi-objective fitness. Such extensions touch few parts of the code. Fitness is a single function that scores an aggregated model on the validation data, so a user-defined objective, for example a weighted combination of metrics or a penalty on tree size, requires editing that function alone. Multi-objective search needs a tuple-valued fitness and one of DEAP's Pareto selection operators in place of their single-objective counterparts. DEAP supplies tree representation, genetic operators, and compilation behind a thin interface, so substituting another GP engine leaves the federated loop, the PyTorch primitives, and the artifact format unchanged.

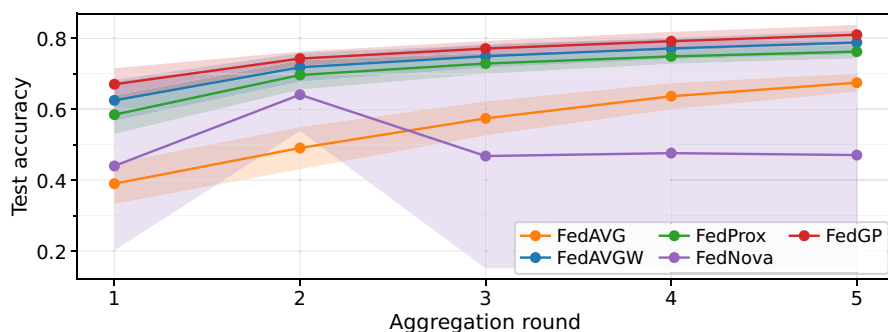
5 Demonstration

To illustrate the intended use of FASTER, we compare all aggregation methods currently available in the platform on FashionMNIST [21]. To the best of our knowledge, this is the first evaluation of FedGP in a setting where clients both drop out of and join the federation during training, a regime that stresses fixed aggregation rules because the participating client set changes across rounds. All methods were run within a single FASTER scenario on one workstation (Intel Core i7-6850K, 64 GB RAM, NVIDIA GTX 1080 Ti and GTX 1080, Ubuntu 20.04), sharing identical dataset partitions, client schedules, model architecture, optimizer settings, evaluation protocol, and seeds. All the parameters presented in this section are customizable from the web application. FashionMNIST was partitioned across 20 clients under a non-IID assumption (imbalance rate 0.8), with 10 clients initially eligible and churn enabled (per-round dropout probability 0.3, new-client probability 0.7); 10% of the training data was reserved server-side before client partitioning and used for the two-epoch warm-up. The validation split is separate from this server-side warm-up data and is used for FedGP fitness evaluation, whereas the test set is loaded independently and used only for round-by-round and final reporting, not for FedGP fitness or model selection. All aggregation methods, including the fixed-rule baselines, received the same warm-up procedure before entering the method-specific aggregation loop. The model is the platform's default neural network. Each client trained for 50 local epochs, sending weights every 10 local epochs and yielding 5 aggregation rounds. FedProx used $\mu = 1$, and FedGP evolved 100 individuals over 20 generations per round (100 generations over the whole run), using the subset of F comprising the four arithmetic operators, together with absolute value, protected square root, mean, and median. Every configuration was repeated 10 times (seeds 42–51). Because all methods share these seeds, they also share the same client-churn sequence in each repetition. Therefore, comparisons are paired at the level of individual churn realiza-

Table 1 Final global-model performance on FashionMNIST under a non-IID scenario with client churn, as mean $\pm$ standard deviation over 10 paired repetitions. F1 is macro-F1 and ROC-AUC is macro one-vs-rest ROC-AU

Method	Accuracy	F1	ROC-AUC	Time (s)
FedGP	0.811 $\pm$ 0.028	0.805 $\pm$ 0.031	0.978 $\pm$ 0.005	817 $\pm$ 85
Weighted FedAvg	0.789 $\pm$ 0.033	0.782 $\pm$ 0.037	0.973 $\pm$ 0.007	249 $\pm$ 6
FedProx	0.763 $\pm$ 0.019	0.753 $\pm$ 0.020	0.967 $\pm$ 0.005	259 $\pm$ 4
FedAvg	0.675 $\pm$ 0.026	0.651 $\pm$ 0.035	0.946 $\pm$ 0.003	249 $\pm$ 3
FedNova	0.471 $\pm$ 0.320	0.427 $\pm$ 0.354	0.779 $\pm$ 0.240	247 $\pm$ 3

Bold values indicate the best results

**Fig. 4** Global-model test accuracy across the five aggregation rounds (mean and standard deviation over 10 repetitions). The FedNova drop reflects collapses observed in 4 of the 10 repetitions under this churn scenario

tions, which supports paired tests and reduces the risk of attributing differences to different participation dynamics.

Table 1 reports final test-set performance, and Fig. 4 shows the evolution of test accuracy across rounds. FedGP attains the best mean on every metric and ranks first in all 10 repetitions, improving mean accuracy by 2.2 percentage points over weighted FedAvg, 4.8 over FedProx, and 13.6 over FedAvg. Using the 10 paired repetitions as blocks, a Friedman test rejects method equivalence for accuracy ($\chi^2 = 35.9$, $p \approx 3 \times 10^{-7}$; mean ranks: FedGP 1.00, weighted FedAvg 2.00, FedProx 3.20, FedNova 4.30, FedAvg 4.50). Paired comparisons of each baseline against FedGP (paired t -test and Wilcoxon signed-rank, Holm–Bonferroni corrected over the four comparisons) confirm significant accuracy gains over all four baselines, with FedGP winning every one of the 10 paired repetitions against every method. Effect sizes are very large for weighted FedAvg, FedProx, and FedAvg (Cohen’s d_z of 2.4, 4.0, 4.4; all corrected $p < 10^{-4}$). For FedNova the paired difference is more variable, because FedNova collapses to a near-trivial predictor in 4 of the 10 repetitions ($d_z = 1.0$); the gain nonetheless remains significant under both tests (Wilcoxon corrected $p = 0.008$, t -test corrected $p = 0.013$).

The shared-churn design also reveals a clear stability difference: FedNova collapsed to a trivial predictor (accuracy ≈ 0.10) in 4 of 10 repetitions, while no other method collapsed. Because the churn sequence is identical across methods, these

collapses appear specific to FedNova's behavior in this scenario rather than to harder churn realizations. FedGP remains about $3.3\times$ slower than fixed-rule baselines (817 s versus ~ 250 s per run), but this is a substantial reduction from the roughly tenfold overhead of earlier implementations, enabled by GPU-based batch evaluation of candidate individuals.

6 Limitations and future work

FASTER targets controlled FL experimentation rather than distributed deployment. Execution is centralized and synchronous within a single infrastructure, so scalability depends on the host environment, and network-level effects (latency, bandwidth, packet loss, asynchronous delivery) are not modeled. Asynchronous, hierarchical, and decentralized topologies are also out of scope. Security evaluation is not yet supported, and there are no modules for secure aggregation, differential privacy, or adversarial defenses. The demonstration is limited to one dataset, one model, and one churn/non-IID scenario, so broader benchmarking is needed before generalizing the observed ranking. FedGP also assumes access to a server-side validation set for fitness evaluation, which may be unavailable in stricter FL deployments with no labeled server-side data. On the platform side, future work will extend FASTER toward distributed, communication-aware execution, security-oriented evaluation, and broader customization. On the evolutionary side, residual runtime overhead relative to fixed-rule baselines remains the principal limitation; surrogate fitness approximation is a natural direction for reducing this gap.

7 Conclusion

This letter presents, to the best of our knowledge, the first public release of FedGP as part of FASTER, an open-source platform for reproducible FL experimentation. The release extends earlier research prototypes in both functionality and efficiency, and removes two barriers that have limited GP-based aggregation research: the lack of a reusable implementation and the difficulty of comparing aggregation rules under identical controlled conditions. By holding all other components fixed, FASTER isolates the effect of the aggregation rule. The demonstration applies this design to what is, to the best of our knowledge, the first FedGP evaluation under client churn, where it consistently outperformed fixed baselines. The remaining cost is runtime overhead, about three times that of fixed-rule baselines, down from roughly ten times in earlier implementations. With this release, FedGP becomes a research object that the evolutionary computation community can inspect, reproduce, and extend.

Acknowledgements This work was partially supported by the HES-SO Jeunes Chercheurs (FASTER grant, ref. 140831), the Spanish Ministry of Economy and Competitiveness (PID2023-147409NB-C22), funded by MCIN/AEI/10.13039/501100011033. This activity was 85% co-financed by the European Union, the European Regional Development Fund, and the Regional Government of Extremadura; Managing Authority; Ministry of Finance, Project GR24142.

Author contributions E.P., F.F.V. and D.C. contributed to the conceptualization of the study. E.P. and V.C. contributed to software development, formal analysis and interpretation of the results, and writing of the original draft. F.F.V. and D.C. contributed to writing – review and editing, supervision, and funding acquisition.

Data availability This study does not generate new datasets. The demonstration is based on FashionMNIST (<https://github.com/zalando-research/fashion-mnist>), and the platform additionally supports the publicly available benchmark datasets MedMNIST (<https://medmnist.com/>), CIFAR-10 and CIFAR-100 (<https://www.cs.toronto.edu/~kriz/cifar.html>), all obtained from their original public sources.

Declarations

Competing interests Francisco Fernández de Vega is a member of the Editorial Board

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. Arcas, Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics (2017). PMLR
2. P. Kairouz, H.B. McMahan, B. Avent, A. Bellet, M. Bennis, A.N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings et al., Advances and open problems in federated learning. *Found. Trends Mach. Learn.* **14**(1–2), (2021)
3. N. Rieke, J. Hancox, W. Li, F. Milletari, H.R. Roth, S. Albarqouni, S. Bakas, M.N. Galtier, B.A. Landman, K. Maier-Hein, et al. The future of digital health with federated learning. *NPJ Digit. Med.* **3**(1), (2020)
4. Y. Liu, Z. Ai, S. Sun, S. Zhang, Z. Liu, H. Yu, In: Yang, Q., Fan, L., Yu, H. (eds.) FedCoin: A Peer-to-Peer Payment System for Federated Learning. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63076-8_9
5. A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, D. Ramage, Federated learning for mobile keyboard prediction (2019), arXiv preprint [arXiv:1811.03604](https://arxiv.org/abs/1811.03604)
6. F. Granqvist, M. Seigel, R. Van Dalen, A. Cahill, S. Shum, M. Paulik, Improving on-device speaker verification using federated learning with privacy (2020), arXiv preprint [arXiv:2008.02651](https://arxiv.org/abs/2008.02651)
7. W. Nie, L. Yu, Z. Jia, Research on aggregation strategy of federated learning parameters under non-independent and identically distributed conditions. In: 2022 4th International Conference on Applied Machine Learning (ICAML) (2022). <https://doi.org/10.1109/ICAML57167.2022.00016>. IEEE
8. T. Li, A.K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, V. Smith, Federated optimization in heterogeneous networks. In: Dhillon, I., Papailiopoulos, D., Sze, V. (eds.) Proceedings of Machine Learning and Systems, vol. 2 (2020)
9. X. Li, K. Huang, W. Yang, S. Wang, Z. Zhang, On the convergence of FedAvg on non-IID data (2020), [arXiv.org/abs/1907.02189](https://arxiv.org/abs/1907.02189)
10. J. Wang, Q. Liu, H. Liang, G. Joshi, H.V. Poor, Tackling the objective inconsistency problem in heterogeneous federated optimization. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS '20, vol. 33. Curran Associates Inc., Red Hook, NY, USA (2020)
11. P. Blanchard, E.M. El Mhamdi, R. Guerraoui, J. Stainer, Machine learning with adversaries: byzantine-tolerant gradient descent. *Adv. Neural Inf. Process. Syst.* **30** (2017)

12. E. Pacioni, C. Muller, F. Vega, D. Calvaresi, FedGP resilience: a comparative study with standard federated aggregation methods under adversarial scenarios, In: García-Sánchez, P., Díaz Álvarez, J., Murphy, A. (eds.) *Applications of Evolutionary Computation*. Springer, Cham (2026)
13. C.T Dinh, N. Tran, J. Nguyen, Personalized federated learning with moreau envelopes. *Adv. Neural Inf. Process. Syst.* **33** (2020)
14. S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, H.B. McMahan, Adaptive federated optimization. *arXiv preprint arXiv:2003.00295* (2020)
15. E. Pacioni, F. Vega, D. Calvaresi, FedGP: genetic programming for evolutionary aggregation in federated learning with non-IID data, In: *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. (2025) Springer
16. E. Pacioni, C. Muller, F.F. Vega, D. Calvaresi, Genetic programming in federated aggregation: a comparison of FedGP with state-of-the-art methods, In: *Proceedings of International Conference on Agents and Artificial Intelligence (ICAART)*. Springer LNAI, (2025)
17. D.J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K.H. Li, T. Parcollet, P.P.B. Gusmão, N.D. Lane, Flower: a friendly federated learning research framework (2020), *arXiv preprint arXiv:2007.14390*
18. C. He, S. Li, J. So, X. Zeng, et al. FedML: A research library and benchmark for federated machine learning (2020), *arXiv preprint arXiv:2007.13518*
19. F. Lai, Y. Dai, S. Singapuram, J. Liu, X. Zhu, H. Madhyastha, M. Chowdhury, FedScale: benchmarking model and system performance of federated learning at scale, In: *Proceedings of the 39th International Conference on Machine Learning (ICML)*. (2022)
20. J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, B. Ni, Medmnist v2-a large-scale light-weight benchmark for 2d and 3d biomedical image classification. *Sci. Data* 10(1), (2023)
21. H. Xiao, K. Rasul, R. Vollgraf, Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms (2017), *arXiv preprint arXiv:1708.07747*
22. A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images (2009)
23. F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: evolutionary algorithms made easy. *J. Mach. Learn. Res.* 13, (2012)
24. A. Paszke, S. Gross, F. Massa, et al. PyTorch: an imperative style, high-performance deep learning library, In: *Advances in Neural Information Processing Systems* 32, (2019)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Elia Pacioni^{1,2}  · Valerio Crocetti^{1,3}  · Francisco Fernández de Vega²  · Davide Calvaresi¹ 

✉ Elia Pacioni
elia.pacioni@hevs.ch

Valerio Crocetti
valerio.crocetti@hevs.ch

Francisco Fernández de Vega
fcofdez@unex.es

Davide Calvaresi
davide.calvaresi@hevs.ch

¹ HES-SO Valais-Wallis, 1950 Sion, Switzerland

² Universidad de Extremadura, 06800 Mérida, Spain

³ Università Politecnica delle Marche, 60121 Ancona, Italy