Adaptive Toggling of Architectural Patterns for Federated Learning

Luciano Baresi
luciano.baresi@polimi.it
Politecnico di Milano
Milan, Italy

Ivan Compagnucci
ivan.compagnucci@gssi.it
GSSI
L'Aquila, Italy

Livia Lestingi
livia.lestingi@polimi.it
Politecnico di Milano
Milan, Italy

Catia Trubiani
catia.trubiani@gssi.it
GSSI
L'Aquila, Italy

Abstract

Federated Learning (FL) is increasingly adopted as an alternative to centralized Machine Learning (ML) techniques, as it allows clients to preserve the privacy of their data. However, FL systems pose new challenges in terms of adaptation, as design choices are conditioned by client characteristics and network conditions, thus necessitating adaptive strategies that elaborate on such a different operational environment. Previous work introduces a set of architectural patterns to support practitioners at design time, but their effectiveness has only been investigated when statically activated throughout the FL process. This work presents a novel FL framework, namely FLiP, where a subset of the aforementioned patterns are dynamically and adaptively toggled in response to evolving performance metrics and boundary conditions. We empirically evaluate FLiP across multiple federation configurations and two learning tasks, considering both static and dynamic conditions. Results indicate that dynamically toggling architectural patterns can be beneficial under specific conditions, with cases leading to an improvement of up to 10% in learning accuracy, at the cost of negligible overhead at deployment time.

Keywords

Federated Learning, Architectural Patterns, Adaptive Pattern

ACM Reference Format:
Luciano Baresi, Ivan Compagnucci, Livia Lestingi, and Catia Trubiani. 2026. Adaptive Toggling of Architectural Patterns for Federated Learning. In *21st International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '26), April 13–14, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Federated Learning (FL) is a distributed machine learning paradigm that ensures data privacy by allowing multiple decentralized clients to collaboratively train a global model without sharing raw data [20, 24]. Introduced by Google in 2016 [29], FL shifts computation locally, where a group of clients trains a Machine Learning (ML) model and sends only the trained parameters to a central server, which aggregates them into a global model.

Typically, the architecture of the FL system is configured *ex ante*, which means that the parameters and architecture design are defined before system deployment and remain fixed [41]. In contrast, *dynamic* FL [33] moves configurations at runtime to improve the efficiency of the learning process [8].

A systematic literature review has recently identified the lack of design approaches and proposed a collection of 14 architectural patterns, organized into 4 categories, to overcome the problem and provide commonly reusable design solutions [26]. Their effectiveness has been evaluated when kept statically active throughout the FL process [3, 9, 10], but more dynamism would allow maximizing various quality metrics. Each pattern entails a trade-off between different quality metrics [27], which is not fully explored when they are statically active or inactive throughout the entire FL process. Consider, for instance, a retail company that trains a demand forecasting model across geographically distributed stores and adopts FL to avoid centralizing sensitive sales data. In the early training phases, involving many clients (e.g., stores) increases both the volume and diversity of training data, which can improve model generalization, while excluding slower clients later on can speed up convergence. Similarly, strategies such as data rebalancing and communication compression are beneficial only under specific conditions (e.g., skewed data distribution, congested network), motivating the adaptive toggling of architectural patterns at runtime.

Inspired by previous work [4, 11, 17], which demonstrated the benefits of self-adaptive FL systems, this paper proposes to dynamically and adaptively toggle FL architectural patterns at runtime. The system dynamically adapts architectural design choices —in a consistent way— to improve the effectiveness of FL. The general idea is exemplified by selecting three patterns as representative of different categories, specifically: (i) client-selector, which is a *client management* pattern; (ii) heterogeneous-data-handler, which is a *model training* pattern; and message-compressor, which is a *model management* pattern. We do not consider the fourth group, since it has a more radical impact on architecture (e.g., hierarchical or decentralized aggregation) that cannot be dynamically (de)activated.

This work proposes the adoption of a runtime controller that adaptively toggles architectural patterns as the FL system evolves. Treating FL evolution as a primary concern, it uses performance indicators and boundary conditions to determine at runtime which architectural solutions are most suitable. The controller is the key component of FLiP, a novel framework that augments the reference FL architecture to toggle architectural patterns at the beginning of each FL round. Unlike previous work with static (de)activation, in each round, FLiP decides which patterns to switch based on system performance metrics and boundary conditions. The former provides quantitative insights on how the FL application is performing and

SEAMS '26, Rio de Janeiro, Brazil

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

whether this requires adjustments; the latter represents a proxy for sources of uncertainty in the environment where the FL application is deployed. Because each pattern comes with a trade-off in terms of metrics it improves or worsens, FLiP digs into this trade-off by identifying regions of the FL design and performance space where activating a pattern is preferable to disable it.

FLiP builds on top of Flower [5], a well-consolidated FL framework. The experimental validation of FLiP addresses three research questions assessing the effectiveness of the framework, its offline and online computational overhead. The results show that the proposed framework achieves a statistically significant performance gain of up to 10%. While the evaluation necessarily covers a limited set of datasets and models, it helps assess the practical benefits of adaptive architectural pattern toggling. In summary, the main contributions of this paper are:

- The development of a framework that automates the application of adaptive strategies to FL systems;
- The exemplification of adaptive strategies for three exemplar architectural patterns from different categories;
- An empirical assessment of the proposed framework under representative FL settings.

In the following, Section 2 outlines the preliminaries, Section 3 describes FLiP, Section 4 reports the experimental results, Section 5 surveys related work and Section 6 concludes the paper.

2 Preliminaries

This section describes the preliminaries that underlie our work.

2.1 Federated Learning in a Nutshell

Federated Learning (FL) enables collaborative training of ML models on data distributed across different devices (the clients), while preserving the privacy of the data of each client. Training occurs through interactions between clients and a server. The process is iterative and lasts multiple rounds until a predetermined stopping criterion holds (e.g., a predefined number of rounds has ended). Initially, ① a central server broadcasts the global model parameters (e.g., the model weights) to all participating clients. Upon receiving these parameters, ② each client independently and locally trains the model using its private dataset, without sharing any raw data with the server. After completion of the local training, ③ each client returns the updated model parameters to the central server. The server then ④ aggregates these local updates, typically using an information fusion algorithm such as FedAvg [29] to ⑤ update the global model. Subsequently, ① the newly aggregated global model parameters are redistributed to the clients, initiating the next training round.

2.2 FL Architectural Patterns

Architectural patterns offer reusable solutions for common design challenges in complex systems [32]. Lo et al. [27] propose a set of patterns tailored to FL, addressing macro areas such as client management, model management, and model training, covering these system dimensions with architectural solutions. In our study, we select: the client-selector, the heterogeneous-data-handler, and the message-compressor. The rationale for this choice is to target different dimensions of FL systems.

Client Selector. The client-selector determines the subset of clients that will participate in the round that is about to begin. Selection criteria can be data-, resource-, or performance-driven [7, 27]. The server profiles each client (e.g., its CPU availability and data distribution) and applies the selection criteria to include or exclude clients.

Potential Impact of Adaptive Toggling. The pattern mitigates issues due to limited resources, imbalanced data, and low-quality local models, enhancing training efficiency and stability. However, having the pattern active for all rounds permanently excludes clients, preventing the system from leveraging additional computational power (i.e., the clients that would be excluded by the selection criteria) even when it could be beneficial.

Heterogeneous Data Handler. The goal of the heterogeneous-data-handler is to mitigate issues due to a client having non-independent and identically distributed (IID) data, which affects the accuracy of the model. To this end, the pattern applies either (i) *data augmentation*, which generates synthetic data to increase the diversity and size of the local dataset through a Generative Adversarial Network (GAN) [16, 39, 44], or (ii) *federated distillation*, which shares knowledge among clients without accessing raw data [27]. These methods balance data distributions while preserving privacy and avoiding centralized data collection [40].

Potential Impact of Adaptive Toggling. While improving the global model accuracy, the heterogeneous-data-handler can cause a surge in computational time to perform data augmentation.

Message Compressor. The message-compressor aims to reduce the communication time overhead by compressing the model parameters exchanged between the server and clients. The process develops in four phases [27]: the server compresses the model weights and sends them to clients; clients decompress the received parameters for local training; after training, clients compress their updated parameters and return them to the server; the server decompresses the incoming updates for aggregation.

Potential Impact of Adaptive Toggling. Weight compression can reduce communication overhead [42]. On the other hand, if the network over which the client and server communicate is fast, compression and decompression only cause additional overhead.

3 The FLiP Framework

The result of an FL run depends on several environmental and system features, which constitute the *semantic space*. We refer to the set of n features in the semantic space as $X \subset \mathbb{R}^n$. Table 1 summarizes the features selected for this study. We refer to the set of m quality metrics selected for the FL system under analysis as $Y \subset \mathbb{R}^m$. The quality metrics selected for this work are listed in Table 1. For example, the number of rounds represents a feature of FL, and the duration of a round r is a quality metric that denotes how long r lasts. Each architectural pattern entails a trade-off between quality metrics, e.g., the heterogeneous-data-handler favors accuracy but comes with a computational cost [27]. The selection of the quality metrics to measure the performance of an FL run derives from an analysis of the requirements of the specific application: in some cases, maximizing the accuracy of the model is the main concern; in other cases, minimizing the training time is preferable [3, 27].

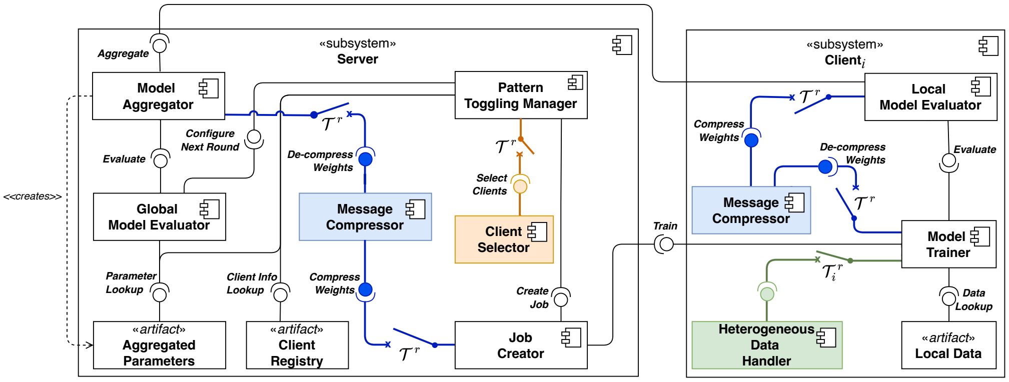


Figure 1: Component diagram representing FLiP's architecture (portions involving different patterns are colored accordingly).

The novelty of FLiP consists in extending the reference architecture by introducing adaptive toggle switches (see Fig. 1) for the selected patterns. When a manager component detects that the current configuration of the semantic space is not the most favorable for metrics Y, adaptation is triggered by flipping a pattern switch (i.e., maintaining a favorable trend for metrics Y is the *adaptation goal*). Specifically, we assume that the federation features a set C of clients in total, while $R \in \mathbb{N}$ represents the budget for FL rounds. Let $\mathbb{B}$ denote the set $\{0, 1\}$. FLiP introduces controllable binary variable $\mathcal{T}_i^r \in \mathbb{B}$ with $i = 1 \ldots |C|$ and $r = 1 \ldots R$ representing whether the selected pattern is active for client i at round r ($\mathcal{T}_i^r = 1$) or not ($\mathcal{T}_i^r = 0$). When the value of $\mathcal{T}_i^r$ is set globally for all clients for round r, we use the simplified notation $\mathcal{T}^r$. Note that the framework currently supports one $\mathcal{T}^r$ variable at a time. Therefore, adaptation involves only one pattern per FL run and the three patterns selected for this study are analyzed independently of each other.

The reference architecture of Lo et al. for FL [26] features two subsystems for the server and the client, whose components reflect the operational workflow described in Section 2.1. Fig. 1 shows the extended architecture with FLiP introducing the Pattern Toggling Manager that plays the key role of determining whether the current architectural setup requires adaptation, i.e., a specific pattern under evaluation needs to be turned on or off. Adaptation decisions in FLiP are managed centrally by the Pattern Toggling Manager on the server side. At the beginning of each FL round, the server communicates the current configuration of the architectural patterns to the clients as part of the round setup. Clients do not perform adaptation decisions autonomously; instead, they react to the received configuration by enabling or disabling the corresponding pattern-specific components. No additional self-adaptation logic nor infrastructure is introduced on the client side.

The value of $\mathcal{T}_i^r$ is the output of a function $t : \mathbb{N}^2 \times \mathbb{R}^n \to \mathbb{B}$, which the Pattern Toggling Manager embeds. The function t takes as input the round and client for which a decision must be made ($i \in [1, |C|]$, $r \in [1, R]$) and the vector $\bar{x} \subseteq \mathbb{R}^n$ of n values representing the current configuration of the semantic space. Function t embeds the *adaptation policy* and $\bar{x}$ contains everything the server knows about the global model, the federation, and the environment in which it is deployed at a specific moment and that can drive the decision to change the toggle state of the pattern. Let $\bar{x}$ and $\bar{x}'$ be space configurations at the beginning of rounds $r-1$ and r, respectively. When $\mathcal{T}_i^{r-1} = t(i, r-1, \bar{x})$ differs from $\mathcal{T}_i^r = t(i, r, \bar{x}')$, $\mathcal{T}_i^{r-1} \to \mathcal{T}_i^r$ is the *adaptation action* for client i at round r.

Table 1: Semantic space in FLiP's problem formulation.

Symbol	Description	$\in X / \in Y$
$R \in \mathbb{N}$	Number of rounds	$\in X$
$RT^r \in \mathbb{R}_+$	Duration of round r	$\in X, \in Y$
C	Available clients	$\in X$
Res_c	Resources on client c	$\in X$
$\mathsf{N}_{\text{high}} \in \mathbb{N}$	High-specification clients	$\in X$
$\mathsf{N}_{\text{low}} \in \mathbb{N}$	Low-specification clients	$\in X$
$C^r \subseteq C$	Clients selected for round r	$\in X$
D_i^r	Data available on client i at round r	$\in X$
$JSD_i^r \in [0, 1]$	JSD score for client i at round r	$\in X$
$CT_i^r \in \mathbb{R}_+$	Communication Time for client i at round r	$\in X / \in Y$
$W_i^r \in \mathcal{M}_{m \times n \times p}(\mathbb{R})$	Weights sent by client i at round r	$\in X$
$\mathrm{F1}^r \in \mathbb{R}_+$	Global F1 score at round r	$\in X / \in Y$
$\mathcal{T}_i^r \in \mathbb{B}$	Pattern toggle state for client i at round r	-

Note that in FLiP, adaptation is managed centrally, which justifies why the manager component is part of the server subsystem. Therefore, the function t can only reason on values that, according to the FL paradigm, the server is allowed to know. For example, adaptation cannot depend on how many samples a client holds for a specific dataset class since the server is not aware of this.

3.1 Adaptation Drivers

Although the tool implementing FLiP allows for a flexible configuration of the semantic space for a FL run, deciding which drivers are most suited to each pattern requires additional reasoning. In the following, we discuss the drivers for the selected patterns.

Client Selector. Out of the alternatives from Section 2.2, in FLiP, we adopt a resource-based selection criterion [27], as per the state of the art [9]. Specifically, we select clients according to their computational power, that is, with a number of CPUs greater than a predefined threshold. We refer to the subset of clients selected for round r as $C^r \subseteq C$ and the amount of resources (e.g., CPUs) available to client i as $Res_i \in \mathbb{N}$. With this selection criterion, client i belongs to set C^r if and only if Res_i is at least $\mathrm{res}_{\mathrm{th}} \in \mathbb{N}$.

In this setting, deactivating the client-selector entails involving the entire federation in the upcoming round rather than filtering out low-specification clients. The function t, therefore, is based on the ratio between the F1 score of the last round and its duration ($\mathrm{F1}^{r-1}/RT^{r-1}$) and the number of high- and low-specification clients available ($\mathsf{N}_{\mathsf{high}}$ and $\mathsf{N}_{\mathsf{low}}$, respectively, such that $\mathsf{N}_{\mathsf{high}} + \mathsf{N}_{\mathsf{low}} = |C|$ holds). At a high level, the manager may decide to deactivate the client-selector if the model performs below expectations, and adding the low-specification clients back in (thus employing additional resources) could improve the situation.

Since it affects the entire federation and not a specific client, the adaptation of client-selector is managed at a global level (i.e., the selection is on with $\mathcal{T}^r = 1$, and off otherwise).

Heterogeneous Data Handler. FLiP applies data augmentation to rebalance non-IID data on a client. When the heterogeneous-data-handler is active for a client, a conditional GAN analyzes the client's distribution and creates class-specific samples accordingly to rebalance the distribution [25]. The pattern thus generates high-quality synthetic data to populate underrepresented classes.

However, these augmentation strategies entail significant time overhead even when classes are only slightly unbalanced (which would thus not affect the accuracy of the global model significantly). Function t is thus based on $\mathrm{F1}^{r-1}$ (if the global model performs worse than expected, it justifies investing more time to improve accuracy) and the degree of imbalance of each client. We measure the deviation of the label distribution of a client from an IID distribution using the Jensen–Shannon Divergence (JSD) metric [18]. Let D_i^r be the data available on client i at round r. We refer to client i's JSD score for round r as $JSD_i^r = JSD(D_i^r) \in [0, 1]$, where $JSD_i^r = 1$ means data distribution is identical to a perfectly balanced one.

In this case, the adaptation is handled client by client ($\mathcal{T}_i^r = 1$ implies that data will be rebalanced on client i at round r).

Message Compressor. Let W_i^r be the model weights at the end of the training on client i at round r. In FLiP, the message-compressor exploits the *LZ77* compression algorithm, known for its fast compression speeds and low resource usage [12], to compress and decompress W_i^r at every client-server communication instance. Note that weight compression is bi-directional: if the message-compressor is active, the server will try to decompress the weights received from all clients and send compressed weights to all clients. Therefore, adaptation is managed at a global level and function t reasons on the duration of the client-server communication phase for the previous round, indicated by $\sum_{i=1}^{|C|} CT_i^{r-1}$ to determine whether activating compression is beneficial.

3.2 Adaptation Policies

This section describes three possible implementations of the t adaptation function, i.e., three adaptation policies for the FLiP framework. We recall that, in our formulation, quality metrics Y depend on observable but partially stochastic factors (e.g., network delays or how data will change on a client in between rounds). In light of this, the adaptation policy that yields the adaptation action cannot be a closed-form optimization problem. FLiP policies, therefore, adopt an expert-driven or fully data-driven approach.

For illustrative purposes, Fig. 2 shows an example of the three different policies for the client-selector as a function of FL round r and the $\mathrm{F1}^{r-1}/RT^{r-1}$ ratio. More details follow hereafter.

Fixed Rule ($\mathrm{FLiP_{rule}}$). The first policy involves making adaptation decisions based on predetermined rules. Rules can imply the comparison of an $\bar{x}_j, j = 1 \ldots n$ driver with a fixed threshold (for example, the JSD score exceeding a critical value) or the comparison between two (or more) drivers, e.g., the global F1 score showing a decreasing trend over the last two rounds.

The selected rule sets for the three patterns are as follows:

$$\mathcal{T}_i^r = \begin{cases} \mathrm{F1}^{r-1} > \mathrm{F1}^{r-2} \wedge \frac{\mathrm{F1}^{r-1}}{RT^{r-1}} > \frac{\mathrm{F1_{min}}}{\mathrm{RT_{min}}} & \text{(client-selector)} \\ JSD_i^r > \mathrm{JSD_{max}} & \text{(het.-data-handler)} \\ RT^{r-1} > \mathrm{RT_{max}} & \text{(message-compressor)} \end{cases} \tag{1}$$

Note that only the heterogeneous-data-handler rule involves a client-specific driver since it is the only pattern out of the three whose adaptation action may vary from client to client.

Figure 2a shows an example of the output of the function t with a fixed policy for client-selector with $\mathrm{F1_{min}}/\mathrm{RT_{min}} = 0.005$ (approximately an accuracy of 0.4 in 65s). As per Eq.1, the pattern is deactivated whenever $\mathrm{F1}^{r-1}/RT^{r-1}$ is below the set threshold.

Although it is the most computationally lightweight among the three alternatives, its main drawback lies in the presence of one or more hyperparameters (depending on the rule set formulation) whose fine-tuning is entirely expert-dependent.

Predictor-based ($\mathrm{FLiP_{pred}}$). The second policy adopts a data-driven approach by training a ML predictor for metric y. For all rounds r and clients i, given the current state of the system (i.e., vector $\bar{x}$), the predictor provides two estimations of the target metric y: one under the hypothesis that the pattern will be activated ($\hat{y}|\mathcal{T}_i^r$) and one for the opposite case ($\hat{y}|\neg\mathcal{T}_i^r$). To this end, the pairs $\langle \bar{x}, y \rangle$ in the training dataset must cover both situations in which the pattern at hand was on or off to properly inform the predictor.

Function t is thus formulated as follows:

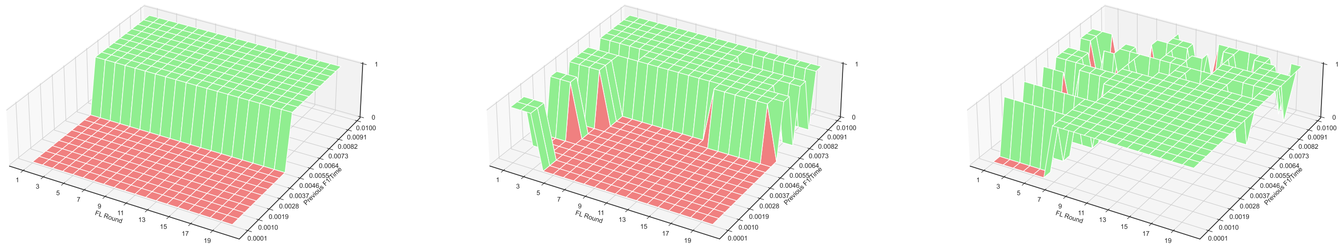
$$\mathcal{T}_i^r = \begin{cases} \hat{y}|\mathcal{T}_i^r > \hat{y}|\neg\mathcal{T}_i^r & \text{(client-sel., het.-data-handler)} \\ \hat{y}|\mathcal{T}_i^r < \hat{y}|\neg\mathcal{T}_i^r & \text{(message-compressor)} \end{cases} \tag{2}$$

With the client-selector and heterogeneous-data-handler, the higher the target metric the better, meaning the pattern is only activated when the predictor yields a higher estimate for that case; vice versa for the message-compressor.

Figure 2b visualizes an example of a predictor-based decision policy. A decision tree regressor is trained with $\langle \bar{x}, y \rangle$ pairs where y is the ratio between the global F1 score and the cumulative duration of the round. Fig. 2b shows the regions where activating the pattern is estimated to be beneficial ($\hat{y}|\mathcal{T}^r > \hat{y}|\neg\mathcal{T}^r$ holds) and vice versa.

Like all data-driven approaches, this policy implies the cost of collecting a sufficiently representative training dataset to train an accurate predictor. In addition, in underrepresented regions of the design or performance space, there remains a risk of reduced accuracy, which can hinder the effectiveness of the policy.

Bayesian Optimization-based ($\mathrm{FLiP_{bo}}$). The third policy also adopts a data-driven approach by framing the per-round decision making process as a Bayesian Optimization (BO) problem [14]. Instead of directly comparing $\hat{y}|\mathcal{T}^r$ and $\hat{y}|\neg\mathcal{T}^r$ as in $\mathrm{FLiP_{pred}}$, this policy envisages an objective function that optimizes an estimate



(a) Fixed policy (FLiP$_{\text{rule}}$). **(b) Predictor-based policy (FLiP$_{\text{pred}}$).** **(c) BayesianOpt-based policy (FLiP$_{\text{bo}}$).**

Figure 2: Policy examples on the client-selector. The z-axis shows t's output as a function of the current round's index and the previous round's performance.

$\hat{y}$ based on $\bar{x}$. To this end, a Gaussian Process Regressor (GPR)—a non-parametric Bayesian model—is trained to predict y [37]. The key advantage of the GPR is that it provides not only a point-wise estimate of the target metric, but also an estimate of the uncertainty of the output. The BO problem then uses an acquisition function (i.e., Expected Improvement) that leverages the GPR's point-wise $\hat{y}$, and uncertainty estimates to explore the search space and identify the action (i.e., activating the pattern or not) that optimizes y.

Figure 2c shows a policy for the client-selector where $\mathcal{T}^r$ is the outcome of the BO optimization problem. Compared to other policies, the semantic space under consideration is more constrained, showing the impact of introducing an optimization process.

In summary, depending on the selected policy, FLiP may incur a computational overhead not only online (i.e., while the FL process is running) but also offline to train reliable predictors for metric $\hat{y}$. We empirically compare the three policies and the (offline and online) costs they imply through experimental validation.

4 Experimental Validation

Our experimentation addresses the following research questions:

RQ1. *How effective is FLiP?*
We quantify the benefit of adopting FLiP and inform software engineers of the improvement they can envisage when deploying it for their applications.

RQ2. *What is the offline computational overhead of FLiP?*
We investigate the offline cost of FLiP to inform software engineers of the initial computational efforts expected.

RQ3. *What is the online computational overhead of FLiP?*
We investigate the online cost of FLiP to advise software engineers about the latencies caused by adaptation.

4.1 Design of the Evaluation

4.1.1 Evaluation Subjects. The experimental evaluation involves two ML tasks each running for 20 FL rounds. The former is a text classification task, where we employ a feed-forward Multi-Layer Perceptron (MLP) trained on the AG_NEWS dataset[1], which contains 120 000 training and 7 600 test news articles labeled by topical categories (e.g., World, Sports, Business). The latter is an image classification task, where we use a Convolutional Neural Network (CNN) trained on the CIFAR-10 dataset[2], comprising 50 000 training and 10 000 test color images of size 32×32 pixels, distributed across animals and object (e.g., airplane, bird, cat).

Table 2: Evaluation subjects features.

Feature	**Value**
Model under training	{MLP, CNN}
Training dataset	{AG_NEWS, CIFAR-10}
FL rounds	20
Memory limit per client	4GB
CPUs per low-spec client	1
CPUs per high-spec client	2
N. High-spec clients ($\mathsf{N}_{\mathsf{high}}$)	{2, 3, 4, 5, 8, 10}
N. Low-spec clients ($\mathsf{N}_{\mathsf{low}}$)	{2, 3, 4, 5, 8, 10}
Data distribution type	{IID, non-IID}
Client data inflow	{one-shot, batched}
Network condition	{stable, unstable}

For the heterogeneous-data-handler, the data augmentation technique depends on the nature of the data. For images, we utilize a Deep Convolutional GAN [31], which comprises a generator and a discriminator. For textual data (e.g., AG_NEWS), we implement a Sequence-Generation GAN [39] utilizing an LSTM generator to convert a latent vector into a sequence of discrete text tokens.

The experiments feature 80 evaluation subjects for each ML task. Table 2 summarizes the variation that we consider for the federation topology and how data are distributed across clients. Each federation has $\mathsf{N}_{\mathsf{high}}$ high-spec clients with 2 CPUs and $\mathsf{N}_{\mathsf{low}}$ low-spec clients with 1 CPU. The selected setups span from a minimum of 6 clients (intended as $\mathsf{N}_{\mathsf{high}} + \mathsf{N}_{\mathsf{low}}$) to a maximum of 20. Subjects also differ according to whether clients have IID data.

The experimental setup additionally explores varying operational conditions, specifically different client data inflow models and diverse network conditions. In the first case, local data are either fully accessible to clients at round 1 (one-shot data inflow) or a

[1] https://huggingface.co/datasets/sh0416/ag_news

[2] https://www.cs.toronto.edu/~kriz/cifar.html

new batch is added at each round (batched data inflow). The dynamics behind clients gathering new data depend on a wide spectrum of uncontrollable factors; given the infeasibility of a universal model, we account for this stochasticity by randomizing the size of the new data batch. Concerning the network, we consider setups with stable or unstable network conditions, i.e., injecting (randomly for the unstable case) delays mimicking network congestion problems [7].

4.1.2 Evaluation Methods. We compare three different policies for FLiP (i.e., $\text{FLiP}_{\text{rule}}$, $\text{FLiP}_{\text{pred}}$, and FLiP_{bo}), each applied to three architectural patterns (i.e., client-selector, the heterogeneous-data-handler, and the message-compressor).

Each policy is compared with three baselines, for a total of 6 methods. For each pattern, the three baselines are: (1) the pattern is never active (never); (2) the pattern is randomly switched on (random); (3) the state of the art in evaluating the specific FL pattern [9]. Through baseline (1), the evaluation determines when it is preferable not to activate the pattern at all. Baseline (2) is a necessary neutral reference point to assess the usefulness of FLiP compared to a purely random choice. Baseline (3) is: (i) for the message-compressor, applying model compression always (always); (ii) for the heterogeneous-data-handler, i.e., it applies data augmentation every time the local dataset changes. For the batched data inflow, this amounts to activating the heterogeneous-data-handler in all rounds for all nodes (always), while with the one-shot data inflow, it is activated only for the first round (once).

For the client-selector, baseline (3) is a system with $\text{N}_{\text{high}} + \text{N}_{\text{low}}$ high-spec clients (all-high), adopting the method used in [9]. A simpler baseline where the selection criterion always excludes all low-spec clients does not constitute a fair comparison, as it would result in FLiP running with $\text{N}_{\text{high}} + \text{N}_{\text{low}}$ clients being compared against a baseline with only N_{high} clients. Our chosen baseline determines how dynamically bringing in some low-spec clients with FLiP compares to having all high-spec clients always active.

4.1.3 Statistical Tests. Given the stochasticity of the FL process, we prevent obtaining favorable results by chance by replicating the application of each method to each evaluation subject 10 times.

When comparing FLiP's performance to the baselines, we follow the guideline introduced by Arcuri and Briand [1]. We apply the Mann-Whitney U test to assess statistical significance and Vargha-Delaney's measure to compute the effect size of the difference between samples [35]. We employ the following standard classification for the effect size: small, medium, or large, defined as values greater than 0.55, 0.63, and 0.70, respectively.

4.1.4 Evaluation Testbed. All experiments are performed on a commodity machine running Ubuntu 24.04, equipped with 48 CPUs, 64 GB of memory, and a base clock speed of 2.20 GHz. FLiP is built upon the Flower framework[3] for FL simulation.

4.2 Results

4.2.1 RQ1: Effectiveness.

Setup. Experiments aim to explore the configuration space of evaluation subjects described in Table 2. Table 3 summarizes the experimental setup of each pattern. The data rebalancing performed with the heterogeneous-data-handler is only relevant with non-IID clients; in the other two cases, experiments are replicated with IID and non-IID clients to evaluate how this impacts FLiP's effectiveness. The state of the network only affects the message-compressor, which is the only pattern evaluated under stable and unstable conditions (i.e., with stochastic network delays). Similarly, the data inflow model has no impact on the message-compressor, whereas the other two patterns are evaluated with both the one-shot and batched inflow models. All patterns are then evaluated on the same 8 ($\text{N}_{\text{high}}, \text{N}_{\text{low}}$) pairs differing in size of the federation ($\text{N}_{\text{high}}+\text{N}_{\text{low}}$) and $\text{N}_{\text{high}} : \text{N}_{\text{low}}$ ratio, for a total of $8 \times 2 \times 2 = 32$ subjects each for the client-selector and message-compressor, and $8 \times 2 = 16$ subjects for the heterogeneous-data-handler. As per Section 4.1.3, the application of each of the 6 methods under comparison (the 3 policies of FLiP and the baselines) to each evaluation subject is replicated 10 times, for a total of $32 \times 6 \times 10 = 1920$ FL runs for the client-selector and message-compressor, and $16 \times 6 \times 10 = 960$ FL runs for the heterogeneous-data-handler.

Table 3: Setup for RQ1.

Architectural Pattern	Data distribution	Data inflow	Network
client-selector	IID, non-IID	{one-shot, batched}	{stable}
heterogeneous-data-handler	non-IID	{one-shot, batched}	{stable}
message-compressor	IID, non-IID	{one-shot}	{stable, unstable}

Building on prior works [9, 27], we expect that patterns favor different performance metrics: client-selector is expected to favor the accuracy-to-training time ratio; heterogeneous-data-handler is expected to favor accuracy; message-compressor is expected to favor client-server communication times. The effectiveness of FLiP compared to the baselines is then evaluated based on metric $y_i(j)$ whose expression varies depending on the pattern ($i \in \{1, 2, 3\}$):

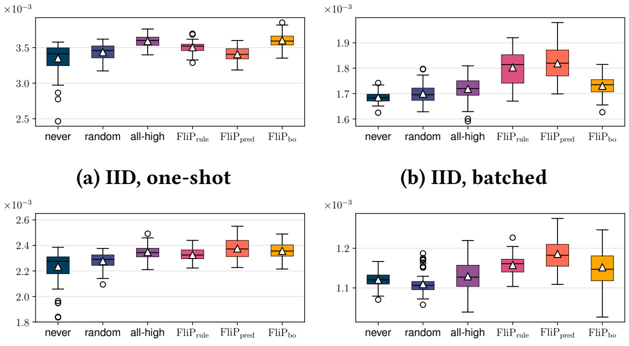
$$y_1(j) = \frac{\text{F1}^j}{\sum_{r=1}^{j} RT^r},\ y_2(j) = \sum_{r=1}^{j} \text{F1}^r,\ y_3(j) = \sum_{r=1}^{j} (\sum_{i=1}^{|C|} CT_i^r) \quad (3)$$

Specifically, for the client-selector, $y_1(R)$ represents the ratio between the final model's accuracy over the cumulative time to train it over R rounds; for the heterogeneous-data-handler, $y_2(R)$ corresponds to the cumulative model's accuracy; for the message-compressor, $y_3(R)$ corresponds to the cumulative communication time. As per Section 4.1.3, we compute the statistical significance of the difference between y_i obtained through FLiP with that obtained through a baseline with the Mann-Whitney U test, and the effect size with Vargha-Delaney's measure.

Results. For the client-selector, y_1 distributions are shown in Fig. 3 and Fig. 4, while Table 4 report the statistical tests results.

The results show that, on the client-selector and with the text classification's task, FLiP's effectiveness is more significant when data is ingested in batches. With image classification, instead, FLiP is more effective with IID clients than in non-IID conditions, and with a one-shot rather than batched data inflow. In general, $\text{FLiP}_{\text{pred}}$ is the best-performing policy for both learning tasks. In all cases, at least one policy performs statistically better than never and random and is at least statistically equivalent to all-high. We recall that all-high employs $\text{N}_{\text{high}} + \text{N}_{\text{low}}$ high-spec clients for all rounds, while FLiP works with N_{high} high-spec clients active at all rounds and

[3] https://flower.ai



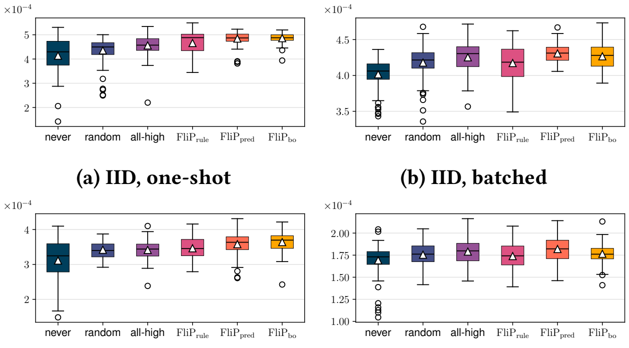
(a) IID, one-shot

(b) IID, batched

(c) non-IID, one-shot

(d) non-IID, batched

Figure 3: $y_1(R)$ on text classification (the higher the better).



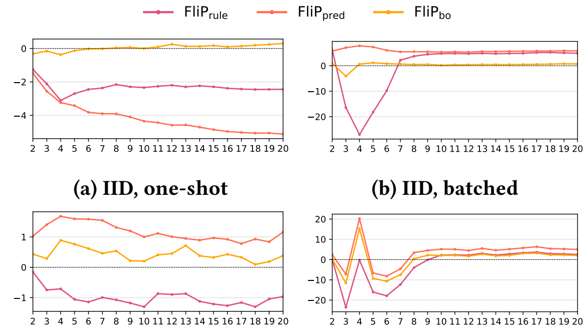
(a) IID, one-shot

(b) IID, batched

(c) non-IID, one-shot

(d) non-IID, batched

Figure 4: $y_1(R)$ on image classification (the higher the better).



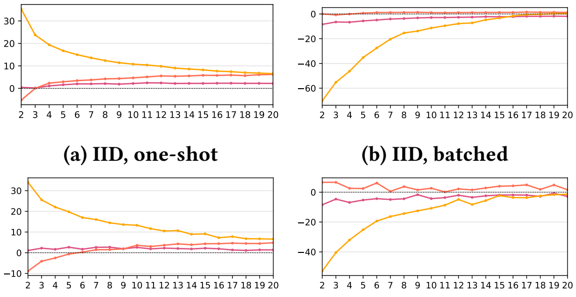
(a) IID, one-shot

(b) IID, batched

(c) non-IID, one-shot

(d) non-IID, batched

Figure 5: Average y_1 per-round percentage change w.r.t. to baseline all-high on text classification (the higher the better).



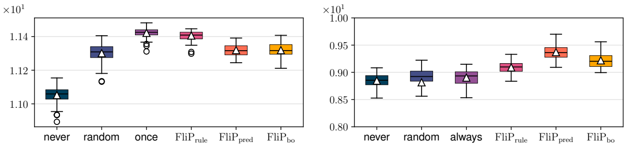
(a) IID, one-shot

(b) IID, batched

(c) non-IID, one-shot

(d) non-IID, batched

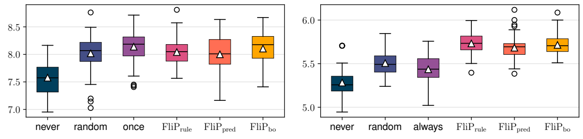
Figure 6: Average y_1 per-round percentage change on image classification (the higher the better).



(a) non-IID, one-shot

(b) non-IID, batched

Figure 7: $y_2(R)$ on text classification (the higher the better).



(a) non-IID, one-shot

(b) non-IID, batched

Figure 8: $y_2(R)$ on image classification (the higher the better).

N$_{low}$ low-spec clients that are adaptively activated. Therefore, a FLiP policy being statistically equivalent to all-high means that it achieves comparable performance with fewer resources, showing a comparative advantage of adaptation.

Figures 5 and 6 show the mean percentage change in metric y_1 with respect to the always baseline for each round, calculated for the three policies of FLiP and the all-high baseline. Specifically, referring to the baseline $y_1(r)$ value as $\bar{y}_1(r)$, the mean percentage change is calculated as the average of $(y_1(r) - \bar{y}_1(r))/\bar{y}_1(r) \cdot 100$ for all the collected data points. This zoomed-in analysis shows that, with the one-shot data inflow, the impact of adaptation is already noticeable at the beginning of the FL process. Instead, with batched data, dynamically toggling the client-selector can be detrimental to performance in the early rounds and progressively increases until it becomes equivalent to the baseline in the final rounds. The explanation for this phenomenon is that, as data availability decreases (as in the initial rounds), it becomes more critical to exclude clients by dynamically toggling client-selector; as the volume of training data stabilizes, the benefits of adaptation become more evident.

For the heterogeneous-data-handler, the distribution of y_2 for all methods is shown in Fig. 7 and Fig. 8 and the results of the statistical tests are reported in Table 5.

The results show that FLiP does not perform statistically better than random and once with the one-shot inflow. This confirms the intuition that, if local datasets are stationary throughout the FL process, applying the heterogeneous-data-handler again after the first round does not have a beneficial impact on performance.

On the other hand, with the batched data inflow, all FLiP policies perform statistically better than all baselines, with FLiP$_{pred}$ and

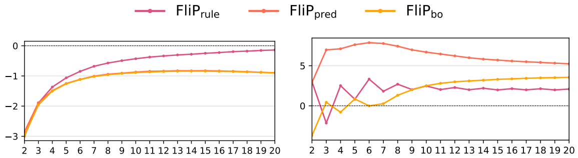
Table 4: Statistical tests results on $y_1(R)$ (L=Large, M=Medium, S=Small, N=Negligible effect size). For each pair, the first row reports IID results and the second reports non-IID results. Cases where FLiP performs better or worse than a baseline are reported in bold and grey, respectively.

Text Classification

	one-shot			batched		
	never	random	all-high	never	random	all-high
FLiP$_{\text{rule}}$	**<0.05 (L)** **<0.05 (L)**	**<0.05 (M)** **<0.05 (M)**	<0.05 (L) <0.05 (S)	**<0.05 (L)** **<0.05 (L)**	**<0.05 (L)** **<0.05 (L)**	**<0.05 (L)** **<0.05 (L)**
FLiP$_{\text{pred}}$	0.45 (N) **<0.05 (L)**	0.06 (S) **<0.05 (L)**	<0.05 (L) **<0.05 (S)**	**<0.05 (L)** **<0.05 (L)**	**<0.05 (L)** **<0.05 (L)**	**<0.05 (L)** **<0.05 (L)**
FLiP$_{\text{bo}}$	**<0.05 (L)** **<0.05 (L)**	**<0.05 (L)** **<0.05 (L)**	0.62 (N) 0.24 (N)	**<0.05 (L)** **<0.05 (M)**	**<0.05 (M)** **<0.05 (L)**	0.08 (S) **<0.05 (S)**

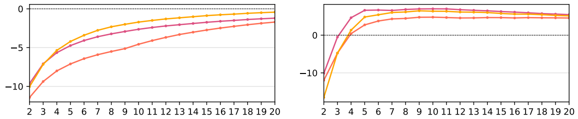
Image Classification

	one-shot			batched		
	never	random	all-high	never	random	all-high
FLiP$_{\text{rule}}$	**<0.05 (M)** **<0.05 (S)**	**<0.05 (M)** 0.35 (N)	0.08 (S) 0.39 (N)	**<0.05 (M)** 0.24 (N)	0.68 (N) 0.61 (N)	<0.05 (S) 0.06 (S)
FLiP$_{\text{pred}}$	**<0.05 (L)** **<0.05 (L)**	**<0.05 (L)** **<0.05 (M)**	**<0.05 (M)** **<0.05 (M)**	**<0.05 (L)** **<0.05 (M)**	**<0.05 (S)** **<0.05 (S)**	0.30 (N) 0.22 (N)
FLiP$_{\text{bo}}$	**<0.05 (L)** **<0.05 (L)**	**<0.05 (L)** **<0.05 (M)**	**<0.05 (L)** **<0.05 (L)**	**<0.05 (L)** **<0.05 (S)**	0.06 (S) 0.68 (N)	0.91 (N) 0.18 (N)



(a) non-IID, one-shot **(b) non-IID, batched**

Figure 9: Average y_2 per-round percentage change on text classification (the higher the better).



(a) non-IID, one-shot **(b) non-IID, batched**

Figure 10: Average y_2 per-round percentage change on image classification (the higher the better).

FLiP$_{\text{bo}}$ showing the best performance in terms of cumulative F1-score on text and image classification, respectively. This shows that re-balancing data multiple times but under selected conditions (i.e., depending on the node's JSD score) results in a statistically higher F1 score than doing it randomly (random) or for all nodes at all rounds (always). However, it is worth to remark that balancing data implies a cost in terms of time overhead quantified in RQ3.

Interestingly, the effectiveness of the heterogeneous-data-handler is driven more by the data inflow model than by the learning task. Despite the different data modalities and augmentation techniques, both tasks mostly benefit from adaptation when data evolves over time, showing how architectural adaptation responds to system dynamics and task-specific characteristics.

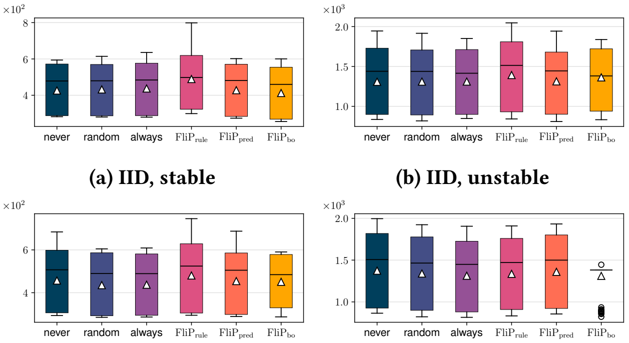
Table 5: Statistical tests results on $y_2(R)$.

Text Classification

	one-shot			batched		
	never	random	once	never	random	always
FLiP$_{\text{rule}}$	**<0.05 (L)**	**<0.05 (L)**	<0.05 (S)	**<0.05 (L)**	**<0.05 (L)**	**<0.05 (L)**
FLiP$_{\text{pred}}$	**<0.05 (L)**	0.08 (S)	<0.05 (L)	**<0.05 (L)**	**<0.05 (L)**	**<0.05 (L)**
FLiP$_{\text{bo}}$	**<0.05 (L)**	0.06 (S)	<0.05 (L)	**<0.05 (L)**	**<0.05 (L)**	**<0.05 (L)**

Image Classification

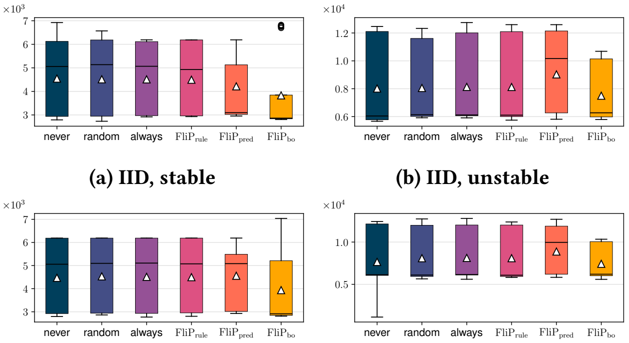
	one-shot			batched		
	never	random	once	never	random	always
FLiP$_{\text{rule}}$	**<0.05 (L)**	0.43 (N)	0.08 (S)	**<0.05 (L)**	**<0.05 (L)**	**<0.05 (L)**
FLiP$_{\text{pred}}$	**<0.05 (L)**	0.22 (S)	0.50 (N)	**<0.05 (L)**	**<0.05 (L)**	**<0.05 (L)**
FLiP$_{\text{bo}}$	**<0.05 (L)**	0.17 (S)	0.42 (N)	**<0.05 (L)**	**<0.05 (L)**	**<0.05 (L)**



(a) IID, stable **(b) IID, unstable**

(c) non-IID, stable **(d) non-IID, unstable**

Figure 11: $y_3(R)$ on text classification (the lower the better).



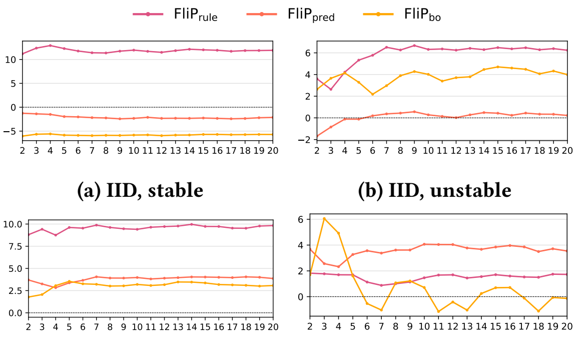
(a) IID, stable **(b) IID, unstable**

(c) non-IID, stable **(d) non-IID, unstable**

Figure 12: $y_3(R)$ on image classification (the lower the better).

These results are further supported by the analysis of the percentage change reported in Fig. 9 and Fig. 10. As discussed, with stationary data, the improvement remains marginal until the end of the FL process. With batched data, instead, the improvement induced by all FLiP's policies emerges early on in the FL process.

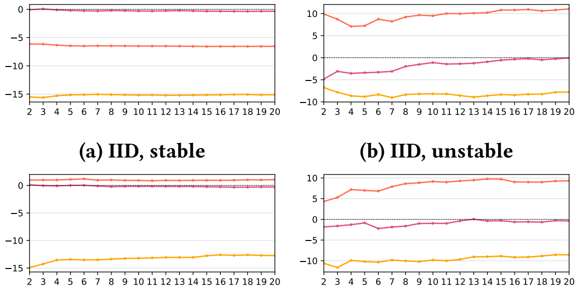
For the message-compressor, $y_3(R)$ distributions are shown in Fig. 11 and Fig. 12, while Fig. 13 and Fig. 14 report the per-round percentage change. The limited size of the model under training also limits the impact that compression has on communication time:



(a) IID, stable **(b) IID, unstable**

(c) non-IID, stable **(d) non-IID, unstable**

Figure 13: Average y_3 per-round percentage change on text classification (the lower the better).



(a) IID, stable **(b) IID, unstable**

(c) non-IID, stable **(d) non-IID, unstable**

Figure 14: Average y_3 per-round percentage change on image classification (the lower the better).

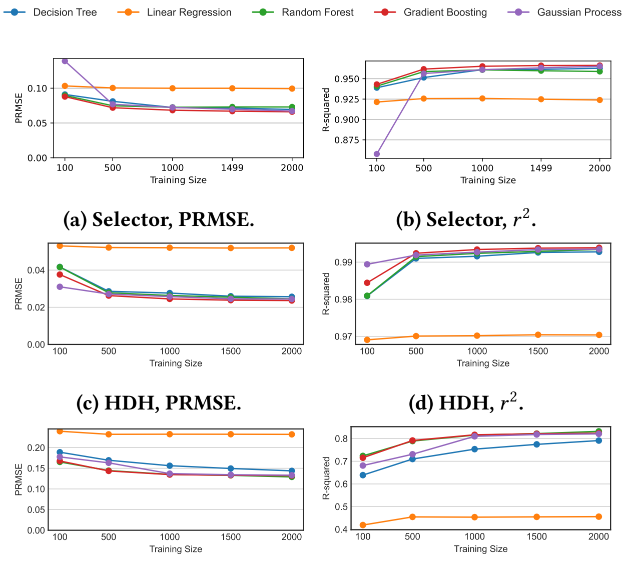
indeed, the three baselines (including the one keeping compression constantly active) and FLiP$_{\text{pred}}$ are equivalent.

For image classification, FLiP$_{\text{bo}}$ consistently leads to a $10-15\%$ average decrease of communication time both in stable and unstable network conditions (Fig. 14). Nevertheless, statistical test results (not reported for the sake of conciseness) do not reveal a significant difference between y_3 distributions (Fig. 12). In summary, experiments highlight a favorable average trend, but with an excessive variance that prevents a clear separation between the distributions.

> **RQ1 Summary.** Adaptive toggling yields a statistically significant 5-10% performance improvement with stationary data for the client-selector and, dually, with variable data for the heterogeneous-data-handler. With the message-compressor, we observe a 10-15% improvement in average, although without statistical evidence related to the adaptation process.

4.2.2 *RQ2: Offline Overhead.*

Setup. The predictor-based (FLiP$_{\text{pred}}$) and BO-based (FLiP$_{\text{bo}}$) policies rely on a predictor trained offline to make decisions at runtime. This research question aims to estimate the volume of training data (i.e., $\langle\bar{x}, y\rangle$ pairs) required to obtain accurate predictors and the time required to collect such data.

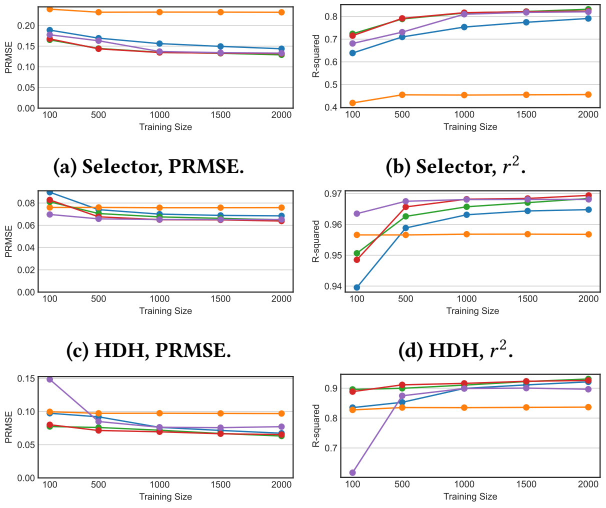


(a) Selector, PRMSE. **(b) Selector, r^2.**

(c) HDH, PRMSE. **(d) HDH, r^2.**

(e) Compressor, PRMSE. **(f) Compressor, r^2.**

Figure 15: RQ2 results for text classification.



(a) Selector, PRMSE. **(b) Selector, r^2.**

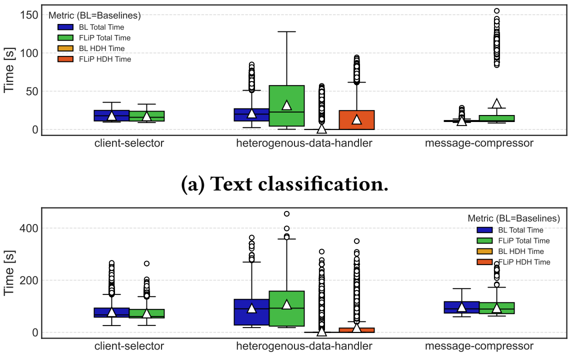
(c) HDH, PRMSE. **(d) HDH, r^2.**

(e) Compressor, PRMSE. **(f) Compressor, r^2.**

Figure 16: RQ2 results for image classification.

For each learning tasks, baseline methods are replicated 10 times on the 80 evaluation subjects, for a total of 2400 FL runs. As per Table 2, each FL run has a budget of 20 rounds, resulting in 19,200 $\langle x, y\rangle$ training samples for the client-selector and message-compressor, and 9,600 for the heterogeneous-data-handler. These data points are collected by applying baseline methods and are, thus, free of any bias that might derive from the application of any FLiP policy.

For each pattern and learning task, we compare five models (Decision Tree, Linear Regression, Random Forest, Gradient Boosting [22], and GPR) with increasing training data. The comparison is



(a) Text classification.

(b) Image classification.

Figure 17: RQ3 results. For heterogeneous-data-handler, we report the time overhead for generating synthetic data separately from the training time.

based on the Percentage Root Mean Square Error (PRMSE) (the lower the better) and r^2 score (the higher the better).

As shown in Fig. 15 and Fig. 16, all regressors perform well in estimating the performance metric selected for all patterns and both learning tasks. With the client-selector and heterogeneous-data-handler, 100 samples (16min for text or 2.5h of computation for images) result in an r^2 score greater than 0.92. With the message-compressor and 1000 samples (2.8h for text or 4.3h of computation for images), all models but the linear regressor have accuracy greater than 0.8. Similarly, PRMSE approximately equals 14% in the worst case. RQ1 experiments adopt a Gradient boosting regressor for text classification and a Decision Tree regressor trained with 2000 samples for image classification. The latter, for the image classification task, for a negligible accuracy cost, is superior to its alternatives in terms of interpretability [30].

RQ2 Summary. FLiP shows the following offline overhead: at most 2.5h of data collection to achieve accuracy ≥ 92% for client-selector and heterogeneous-data-handler, and at most 4.3h for the message-compressor.

4.2.3 *RQ3: Online Overhead.*

Setup. This RQ accounts for the costs sustained at runtime during the FL process. We aim to quantify the computational time required for the adaptation process to run FLiP. To this end, we compare the duration of FL rounds in all experiments performed for RQ2 between FLiP and all baseline methods for the three patterns. For the heterogeneous-data-handler, we also compare the additional overhead necessary to re-balance the data.

Results. For the text classification task and client-selector, as per Fig. 17a, FLiP adds no computational overhead and yields a significant performance boost, indicating that adaptation is beneficial. In contrast, with message-compressor, adaptation incurs significant computation overhead without reducing communication time, suggesting it is not cost-effective for simpler tasks. For image classification (Fig. 17b), using FLiP with client-selector and message-compressor adds no computational overhead, suggesting that adaptation is more beneficial for heavier learning tasks.

Concerning the heterogeneous-data-handler, for both learning tasks, results confirm that performing data augmentation multiple times leads to a statistically significant increase of the duration of the FL round (by 50% for text and 27% for images), counterbalanced by the gain in accuracy discussed in RQ1.

RQ3 Summary. FLiP demonstrates a negligible difference in time overhead when applied to the client-selector and, for image classification, to the message-compressor. A task-dependent 27-50% increase in round duration is found for heterogeneous-data-handler, balanced by an accuracy boost.

4.3 Discussion

We discuss the main findings from the experimental campaign, along with the threats to validity and limitations [38].

4.3.1 *Lessons Learned.* The evaluated adaptation policies exhibit distinct tradeoffs. The expert-driven rule set is a low-overhead baseline but struggles under non-stationary dynamics (e.g., batched scenarios) where optimal policies evolve. The predictor-based policy performs best, particularly with client-selector, indicating that when pattern effects correlate with observable metrics, predictors trained on historical data achieve a favorable performance–cost balance. Bayesian optimization (BO) effectively balances exploration and exploitation in highly uncertain settings (e.g., heterogeneous-data-handler with batched data), at the expense of higher computational cost. Although data collection incurs an upfront cost, it is a one-time investment amortized over the FL system's lifetime.

Adaptation effectiveness is task-dependent. Text classification benefits more from adaptive client selection under evolving data inflow, whereas image classification is more sensitive to data distribution stability and model size. These results support adaptive architectural patterns as task-aware design choices rather than universally optimal solutions.

Although FLiP does not explicitly model training phases, mode-like behavior emerges implicitly, as runtime metrics evolve with model maturity, making different patterns preferable at different stages of the FL process.

Overhead results highlight the practicality of adaptive toggling. With heterogeneous-data-handler, whether accuracy gains justify longer training depends on application priorities (e.g., critical medical settings). For message-compressor, adaptation benefits are limited by small model sizes and often dominated by environmental stochasticity, indicating the need for more context-aware policies.

4.3.2 *Threats to Validity.*

External Validity. To mitigate external validity threats, we conducted an extensive campaign over 80 evaluation subjects, totaling 4,800 FL runs per ML task, with each experiment repeated 10 times and analyzed using Mann–Whitney U tests at a 95% confidence level. FLiP supports analyses across varying federation sizes, FL rounds, and parameter settings to improve generalizability. Nevertheless, the evaluation is limited in terms of datasets, model architectures, and FL patterns; results should therefore be interpreted as evidence of feasibility rather than universal applicability, with broader evaluations left to future work.

Internal Validity. Internal validity threats stem from the choice of numerical input parameters used to assess performance variations across architectural patterns. We vary data distributions (IID vs. non-IID) to evaluate heterogeneous data handling while keeping inputs consistent across experiments to reduce confounding effects. However, selecting representative parameters remains challenging [6], and additional configurations require future investigation.

A methodological limitation of FLiP is that only one architectural pattern can be toggled at a time. Simultaneous pattern activation may introduce configuration conflicts (e.g., reduced data diversity limiting data rebalancing effectiveness), which will be addressed in future work. Security-oriented patterns (e.g., secure aggregation or homomorphic encryption) could also be integrated but are driven by privacy and trust concerns rather than performance trade-offs and are therefore left for future work.

Construct Validity. Construct validity threats arise from potential metric misuse. We report standard classification metrics (e.g., F1) and basic count-based metrics (e.g., total FL round runtime RT_r), minimizing ambiguity.

5 Related Work

This section surveys related work on architectural solutions and self-adaptation approaches in FL systems.

Architectural Solutions for FL. Previous studies analyze the FL paradigm from a software architecture perspective. Zhang et al. [42] compare four alternatives for system architectures of FL: centralized, hierarchical, regional, and decentralized, and analyze their communication overhead, model evolution speed, and overall scalability. *FLRA* [26] is a pattern-oriented reference architecture for FL systems. They derive an end-to-end blueprint, encompassing phases such as job creation, model deployment, and monitoring, by synthesizing both academic research and industrial best practices.

Lo et al. [27] and Di Martino et al. [13] build a catalog of architectural patterns tailored to FL, of which the client-selector is an example. Each pattern maps a specific stage in the FL model life cycle, providing a straightforward blueprint for practitioners. However, while their catalog offers important insights, it mainly serves as a reference for recurring design solutions.

Several studies benchmark FL performance under varying circumstances. Li et al. [24] provide a foundational survey on FL, addressing how constraints such as limited on-device resources, non-IID distributions, and user privacy shape the need for novel optimization schemes. Lai et al. [21] developed FedScale, a FL benchmarking suite featuring realistic datasets and a scalable runtime environment, enabling reproducible FL research. FedScale offers high-level APIs to implement, deploy, and evaluate FL algorithms across diverse hardware and software with minimal effort. Previous work focuses on node selection strategies [15, 28], highlighting selection principles (e.g., data heterogeneity, hardware constraints), scheduling challenges, and research directions.

In summary, foundational studies on FL from a software architecture perspective [26, 27] primarily consist of qualitative analyses of foundational literature, combined with industrial case studies. Our work provides empirical evidence on how architectural decisions taken at runtime affect FL performance. The most recent contribution in this line of research [9] selects, implements, and quantitatively compares three patterns from Lo's catalog [27]. Their results highlight trade-offs among accuracy, latency, and computational efficiency when statically chosen for an entire FL. Our work represents a critical step forward, introducing dynamic pattern activation on a per-round basis.

Self-adaptation in FL. Baresi et al. [4] frame FL as a self-adaptation problem that, based on a target accuracy for the new round, estimates the number of epochs considering clients' resource limits and accuracy from the previous two rounds. Wang et al. [36] propose a control algorithm that determines the best trade-off between local update and global parameter aggregation to minimize the loss function under a given resource budget. Li et al. [23] propose a method where the server leverages historical data to predict the feasible workload for each client. Zhang et al. [43] propose adaptively adjusting the batch size for each client at every round to mitigate performance degradation caused by non-IID data. The work of Ilhan et al. [19] addresses the challenge of down-scaling the model by incorporating "early exit classifiers" to improve the cost-effectiveness of training on resource-constrained clients. Singh et al. [34] develop a FL strategy to address heterogeneous data by applying adaptive masking on unlabeled data, reducing entropy and enhancing confidence of the model's predictions.

While these approaches demonstrate the value of self-adaptation, they primarily focus on fine-tuning low-level training hyperparameters (e.g., epochs, batch size) or the model structure itself and not on adaptation at the architectural level. Our work is the first to bridge this gap by applying self-adaptation principles to the selection and toggling of architectural patterns, a fundamentally different and complementary approach to optimizing FL systems.

6 Conclusion

This paper presents FLiP, a novel framework to adaptively toggle architectural patterns at the beginning of each FL round. We quantify the benefits of FLiP through an experimental campaign on text and image classification tasks, considering both IID and non-IID client data distributions and different system settings (e.g., federation size, client data inflow, and system congestion affecting computation and communication) that emulate realistic FL deployments. Our findings indicate that FLiP can lead to an improvement in learning accuracy and introduces negligible computational overhead, both offline and online, for two of the three selected patterns.

Future work will address the points raised as threats to validity, along with a possible extension of the set of patterns under analysis, thus further investigating the pros and cons of dynamically toggling additional architectural patterns in FL systems.

Data Availability

The replication package is publicly available on Zenodo [2].

Acknowledgments

This work has been partially funded by the PRIN 2022 projects nr. 20228FT78M and P20224K9EK, the PNRR ECS00000041 VITALITY, and Department of Excellence 2023 - 2027 for GSSI, all funded by the Italian Ministry of University and Research (MUR).

References

[1] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Intl. Conf. on Software Engineering*. 1–10.

[2] Luciano Baresi, Ivan Compagnucci, Livia Lestingi, and Catia Trubiani. 2026. Open Science Artifact: Adaptive Toggling of Architectural Patterns for Federated Learning. doi:10.5281/zenodo.17435560

[3] Luciano Baresi, Livia Lestingi, and Iyad Wehbe. 2025. Architecting Federated Learning Systems: A Requirement-Driven Approach. In *Proceedings of the European Conference on Software Architecture (ECSA) (Lecture Notes in Computer Science, Vol. 15929)*. Springer, 224–239.

[4] Luciano Baresi, Giovanni Quattrocchi, and Nicholas Rasi. 2021. Federated machine learning as a self-adaptive problem. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 41–47.

[5] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. 2020. Flower: A Friendly Federated Learning Research Framework. *CoRR* abs/2007.14390 (2020).

[6] André B Bondi. 2015. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice.*

[7] Christopher Briggs, Zhong Fan, and Peter Andras. 2020. Federated Learning with Hierarchical Clustering of Local Updates to Improve Training on Non-IID Data. In *International Conference on Neural Network*. 1–9.

[8] Shunfeng Chu, Jun Li, Jianxin Wang, Zhe Wang, Ming Ding, Yijin Zhang, Yuwen Qian, and Wen Chen. 2022. Federated learning over wireless channels: Dynamic resource allocation and task scheduling. *IEEE Transactions on Cognitive Communications and Networking* 8, 4 (2022), 1910–1924.

[9] Ivan Compagnucci, Riccardo Pinciroli, and Catia Trubiani. 2025. Performance Analysis of Architectural Patterns for Federated Learning Systems. In *International Conference on Software Architecture (ICSA)*. IEEE, 289–300.

[10] Ivan Compagnucci, Riccardo Pinciroli, and Catia Trubiani. 2026. Experimenting Architectural Patterns in Federated Learning Systems. *Journal of Systems and Software* 232 (2026), 112655.

[11] Rustem Dautov and Erik Johannes Husom. 2024. Raft Protocol for Fault Tolerance and Self-Recovery in Federated Learning. In *2024 IEEE/ACM 19th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 110–121.

[12] Peter Deutsch and Jean-Loup Gailly. 1996. ZLIB Compressed Data Format Specification version 3.3. *RFC* 1950 (1996), 1–11.

[13] Beniamino Di Martino, Domenico Di Sivo, and Antonio Esposito. 2024. Architectural Patterns for Software Design Problem-Solving in the Implementation of Federated Learning Structures Within the E-Health Sector. In *International Conference on Advanced Information Networking and Applications*. Springer, 347–356.

[14] Peter I Frazier. 2018. Bayesian optimization. In *Recent advances in optimization and modeling of contemporary problems*. Informs, 255–278.

[15] Lei Fu, Huanle Zhang, Ge Gao, Mi Zhang, and Xin Liu. 2023. Client selection in federated learning: Principles, challenges, and opportunities. *IEEE Internet of Things Journal* 10, 24 (2023), 21811–21819.

[16] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Conference on Neural Information Processing Systems*. 2672–2680.

[17] Chenyu Hu, Mingyue Zhang, Nianyu Li, Jialong Li, Zheng Yang, Muneeb Ul Hassan, and Kenji Tei. 2025. Adapting Aggregation Rule for Robust Federated Learning under Dynamic Attacks. In *2025 IEEE/ACM 20th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 171–177.

[18] Zhiyao Hu, Dongsheng Li, Ke Yang, Ying Xu, and Baoyun Peng. 2025. Optimizing Data Distributions Based on Jensen-Shannon Divergence for Federated Learning. *Tsinghua Science and Technology* 30, 2 (2025), 670–681.

[19] Fatih Ilhan, Gong Su, and Ling Liu. 2023. Scalefl: Resource-adaptive federated learning with heterogeneous clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 24532–24541.

[20] Peter Kairouz et al. 2021. Advances and Open Problems in Federated Learning. *Foundations and Trends in Machine Learning* 14, 1-2 (2021), 1–210.

[21] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. FedScale: Benchmarking Model and System Performance of Federated Learning at Scale. In *Proceedings of Machine Learning Research (PMLR)*. 11814–11827.

[22] Machine Learning. 1997. Tom Mitchell. *McGraw Hill* (1997), 31.

[23] Li Li, Moming Duan, Duo Liu, Yu Zhang, Ao Ren, Xianzhang Chen, Yujuan Tan, and Chengliang Wang. 2021. FedSAE: A novel self-adaptive federated learning framework in heterogeneous systems. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–10.

[24] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine* 37, 3 (2020), 50–60.

[25] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2020. On the Convergence of FedAvg on Non-IID Data. In *International Conference on Learning Representations, (ICLR)*.

[26] Sin Kit Lo, Qinghua Lu, Hye-Young Paik, and Liming Zhu. 2021. FLRA: A reference architecture for federated learning systems. In *European Conference on Software Architecture*. Springer, 83–98.

[27] Sin Kit Lo, Qinghua Lu, Liming Zhu, Hye-Young Paik, Xiwei Xu, and Chen Wang. 2022. Architectural patterns for the design of federated learning systems. *Journal of Systems and Software* 191 (2022), 111357.

[28] Samara Mayhoub and Tareq M. Shami. 2024. A review of client selection methods in federated learning. 31, 2 (2024), 1129–1152.

[29] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, Vol. 54. 1273–1282.

[30] Christoph Molnar. 2020. *Interpretable machine learning*. Lulu. com.

[31] Avik Pal and Aniket Das. 2021. TorchGAN: A Flexible Framework for GAN Training and Evaluation. *Journal of Open Source Software* 6, 66 (2021), 2606.

[32] Mark Richards. 2015. *Software Architecture Patterns*. Vol. 4.

[33] Elsa Rizk, Stefan Vlaski, and Ali H Sayed. 2020. Dynamic federated learning. In *Proceedings of the International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. 1–5.

[34] Neha Singh and Mainak Adhikari. 2025. SelfFed: Self-adaptive Federated Learning with Non-IID data on Heterogeneous Edge Devices for Bias Mitigation and Enhance Training Efficiency. *Information Fusion* 118 (2025), 102932.

[35] András Vargha and Harold D Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[36] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. 2019. Adaptive Federated Learning in Resource Constrained Edge Computing Systems. *IEEE J. Sel. Areas Commun.* 37, 6 (2019), 1205–1221.

[37] Christopher Williams and Carl Rasmussen. 1995. Gaussian processes for regression. *Advances in neural information processing systems* 8 (1995).

[38] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. 2012. *Experimentation in Software Engineering*. Vol. 236.

[39] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *Conference on Artificial Intelligence AAAI*. 2852–2858.

[40] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. 2021. A Survey on Federated Learning. *Knowledge-Based System* 216 (2021), 106775.

[41] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. 2021. A survey on federated learning. *Knowledge-Based Systems* 216 (2021), 106775.

[42] Hongyi Zhang, Jan Bosch, and Helena Holmström Olsson. 2020. Federated learning systems: Architecture alternatives. In *Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 385–394.

[43] Jie Zhang, Song Guo, Zhihao Qu, Deze Zeng, Yufeng Zhan, Qifeng Liu, and Rajendra Akerkar. 2021. Adaptive federated learning on non-iid data with resource constraint. *IEEE Trans. Comput.* 71, 7 (2021), 1655–1667.

[44] Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. 2017. Adversarial Feature Matching for Text Generation. In *International Conference on Machine Learning, ICML*, Vol. 70. PMLR, 4006–4015.