FLOAT: Federated Learning Optimizations with Automated Tuning

Ahmad Faraz Khan*
Virginia Tech
United States
ahmadfk@vt.edu

Azal Ahmad Khan
Indian Institute of Technology, Guwahati
India

Ahmed M. Abdelmoniem
Queen Mary University of London
United Kingdom

Samuel Fountain
University of Minnesota
United States

Ali R. Butt
Virginia Tech
United States

Ali Anwar
University of Minnesota
United States

Abstract

Federated Learning (FL) has emerged as a powerful approach that enables collaborative distributed model training without the need for data sharing. However, FL grapples with inherent heterogeneity challenges leading to issues such as stragglers, dropouts, and performance variations. Selection of clients to run an FL instance is crucial, but existing strategies introduce biases and participation issues and do not consider resource efficiency. Communication and training acceleration solutions proposed to increase client participation also fall short due to the dynamic nature of system resources. We address these challenges in this paper by designing FLOAT, a novel framework designed to boost FL client resource awareness. FLOAT optimizes resource utilization dynamically for meeting training deadlines, and mitigates stragglers and dropouts through various optimization techniques; leading to enhanced model convergence and improved performance. FLOAT leverages multi-objective Reinforcement Learning with Human Feedback (RLHF) to automate the selection of the optimization techniques and their configurations, tailoring them to individual client resource conditions. Moreover, FLOAT seamlessly integrates into existing FL systems, maintaining non-intrusiveness and versatility for both asynchronous and synchronous FL settings. As per our evaluations, FLOAT increases accuracy by up to 53%, reduces client dropouts by up to 78×, and improves communication, computation, and memory utilization by up to 81×, 44×, and 20× respectively.

*Corresponding author.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

EuroSys '24, April 22–25, 2024, Athens, Greece
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0437-6/24/04.
https://doi.org/10.1145/3627703.3650081

CCS Concepts: • **Computing methodologies → Machine learning**; *Distributed algorithms.*

Keywords: Federated Learning, Machine Learning Systems, Resource Management

ACM Reference Format:

Ahmad Faraz Khan, Azal Ahmad Khan, Ahmed M. Abdelmoniem, Samuel Fountain, Ali R. Butt, and Ali Anwar. 2024. FLOAT: Federated Learning Optimizations with Automated Tuning. In *Nineteenth European Conference on Computer Systems (EuroSys '24), April 22–25, 2024, Athens, Greece.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3627703.3650081

1 Introduction

Distributed Machine Learning (ML) workflows typically involve collecting training data at some central location from multiple sources to train a model. However, regulations like GDPR [64] and HIPAA [52], the reluctance of enterprises to share data due to competitiveness [21], liability in collecting personal information from edge devices [5, 17, 60], and high data movement costs prevent such centralized collection of data. To this end, Federated Learning (FL) [49] has emerged as a viable solution, enabling collaborative ML model training across numerous clients without requiring data sharing [3, 49]. Despite the promise of FL in privacy preservation and its successful applications in diverse fields such as consumer devices [24, 78], healthcare [19, 74, 82], finance [6, 47, 63], and manufacturing [31, 55, 88], we are faced with a number of challenges [30, 40] when utilizing FL in practice.

In contrast to traditional ML, which relies on data distributed in an Independent and Identical Distributed (IID) manner, FL environments are inherently heterogeneous stemming from its distributed and diverse client population. This heterogeneity gives rise to numerous challenges such as stragglers [10], dropouts [40], and a decline in model performance due to the intrinsic data and resource variations [1, 30, 40]. When clients fail to meet deadlines or complete training, the progress made is lost [1, 15, 40]. This not only wastes resources allocated to training and communication but also affects overall accuracy. Many IoT, Edge, and mobile devices already operate with constraints in compute, network, energy, and memory [2, 30]. Expending these limited resources carelessly is particularly problematic when devices have caps, like on memory or network usage, or restricted battery and compute times. Moreover, resources set aside for training could have been utilized for other applications, leading to both diminished FL performance and fewer resources available for other tasks. Current strategies either prioritize devices likely to complete local training promptly [2, 34, 39, 65] or adopt asynchronous training for global model updates [2, 35, 51]. Nonetheless, both approaches can induce participation bias in highly heterogeneous scenarios, negatively affecting model performance. Furthermore, client availability for client selection has proven to be pivotal in improving resource efficiency and time to converge [1, 2, 34, 65, 67, 85]. However, availability is considered a fixed linear window of time [2, 34, 65] which is an unrealistic assumption because availability highly depends on the limited network, energy, compute resources, and resource consumption of concurrent running applications which makes the prediction of availability periods a challenging task.

To enhance client retention, several acceleration techniques have been suggested to fine-tune the balance between client involvement and model performance. These encompass quantization [57, 61], compression [73], partial training [83], and model pruning [29, 66, 81]. However, our evaluation has unveiled substantial disparities in both performance and resource efficiency when employing these fixed-configuration acceleration techniques in the context of clients' ever-changing resource conditions. Moreover, acceleration techniques are not interchangeable; each one brings unique acceleration advantages while also influencing model accuracy in different ways. Combining these techniques in a constantly changing, resource-limited environment adds layers of complexity to the selection and configuration process. Furthermore, we have observed that heuristic-based solutions for dynamic acceleration configuration fall short as they cannot be finely tuned for diverse workloads and fail to account for intricate resource fluctuations. As a result, selecting and configuring the appropriate techniques for each training round becomes challenging due to the dynamic nature of resource availability and consumption on client devices. These devices exhibit varying resource characteristics over time, creating a multifaceted environment that cannot be effectively managed using static accelerations or heuristic-based approaches.

In this paper, we present FLOAT, an innovative framework designed to optimize both model performance and resource efficiency. FLOAT achieves this by leveraging a multi-objective Reinforcement Learning with Human Feedback (RLHF) agent, which minimizes stragglers and dropouts. Central to FLOAT's approach is the balance between model performance and resource consumption, allowing clients to make the most of

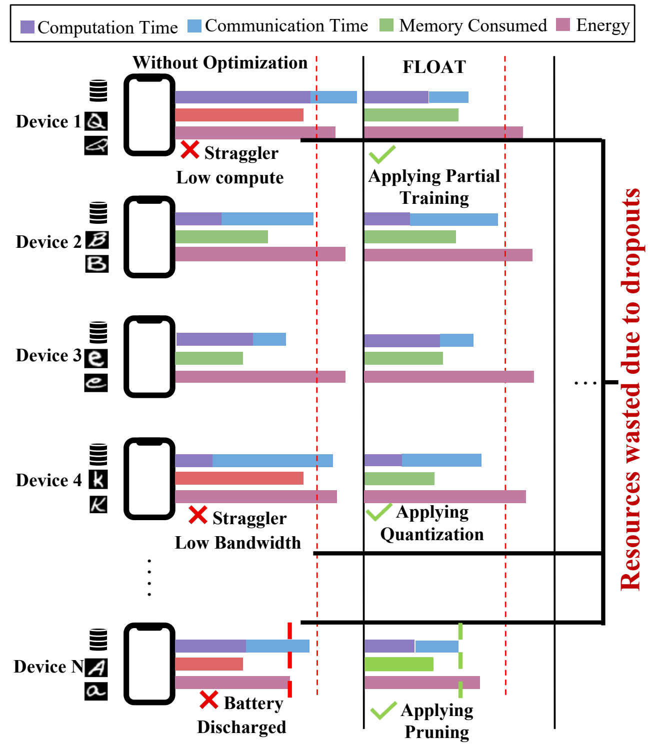


Figure 1. Optimization space of FL is large, considering the scale of decentralized training, system heterogeneity, data heterogeneity, and sources of runtime variance.

their resources to meet training objectives without experiencing dropouts due to staleness. To improve resource efficiency and model performance, and reduce training time, FLOAT incorporates various strategies such as model quantization, partial training, and pruning. Unlike previous works that either depend solely on a single optimization technique [29, 57, 83] or adopt a heuristics-based approach [75], FLOAT uniquely employs an RLHF agent to determine the most suitable optimization method and its configuration. We have also enhanced FLOAT's scalability and minimized its overhead by introducing a Q-learning based RLHF agent that narrows down the states from an otherwise unlimited set of combinations. Furthermore, FLOAT's design is both generalized and reusable due to its ability to fine-tune the RLHF agent at a minimal cost. Importantly, FLOAT is non-intrusive, offering a smooth integration with existing FL systems and client selection algorithms, without affecting the core training procedures.

2 Background

FL is a decentralized ML approach [30]. It enables multiple devices or entities to collaboratively train a shared model without sharing raw data, maintaining data decentralization and privacy. The process involves training a global model

θ on each participating device using local data and then aggregating the model updates to improve the global model.

In each training round, participating devices execute local updates using their local datasets, optimizing their local models θ_i with an optimization algorithm such as stochastic gradient descent (SGD) [58]. This update is governed by the following equation:

$$\theta_i \leftarrow \theta_i - \eta \cdot \nabla L(\theta_i, D_i)$$

Where θ_i represents the local model parameters, η is the learning rate, and $\nabla L(\theta_i, D_i)$ is the gradient of the local loss function $L(\theta_i, D_i)$. After local updates, devices transmit their model updates to the central server, which aggregates them using methods like simple averaging [49]:

$$\theta \leftarrow \frac{1}{N} \sum_{i=1}^{N} \theta_i$$

Where N is the number of participating devices. This process iterates for multiple rounds until convergence criteria are met. Finally, the global model θ is evaluated and can be deployed. This FL method described above is also known as FedAvg [49] and forms the basis of other popular works in FL [2, 34, 39, 51].

FL not only allows training on distributed data but also provides advantages like preservation of data privacy and communication reduction. It is beneficial in situations where data is sensitive or spread over multiple devices, such as in healthcare, finance, and IoT applications [77]. Yet, FL faces issues due to system and data differences across client devices [40].

The data and resource heterogeneity among clients degrades model performance and leads to variable response time among clients (i.e., the time between a client receiving the training task and returning the results) in cross-device FL, which is usually referred to as the straggler problem. As visualized in Figure 1, different devices face distinct challenges, such as low computational capacity, bandwidth limitations, and battery discharges, that can exacerbate this issue. Without recognizing the variable resources at clients and implementing adaptive optimizations, there's a risk of stragglers in FL compromising the system's efficiency, prolonging convergence time, and degrading resource utilization. In the worst case, clients with limited resources cannot train any further after exhausting their available resources and are unable to respond back with updates and are considered **dropouts** [2, 16, 40].

3 Related Work

Recent research has largely addressed statistical heterogeneity due to non-IID data distributions [12, 23, 33, 42, 56, 68]. However, system heterogeneity remains less explored. In one set of studies, strategies for intelligent client selection are proposed, as seen in Harmony [65], Oort [39], PyramidFL [25], FedProx [41], and other related works [22]. However, these works assume static and consistent resource availability at clients and their performance decreases with increased heterogeneity. Some approaches use Reinforcement Learning (RL) for client selection. AutoFL [34] focuses on energy efficiency, while MARL [43] aims to improve accuracy. However, these approaches don't provide a complete solution to FL's heterogeneity as they address specific aspects. Another line of research suggests asynchronous communication methods [9, 11, 35, 51] focusing on quick learning by using client resources extensively. This allows slower clients to continue training with older models and contribute to aggregation later. However, this introduces challenges like reduced accuracy, client selection bias, and slower convergence. Other research focuses on acceleration techniques to reduce client-side computational and communication costs, such as quantization [57, 61], compression, partial training [83], and model pruning [29, 81], aiming to increase client involvement.

Selecting the right techniques for each training round is challenging because of the changing nature of edge device resources. Client availability plays a key role in resource efficiency and convergence time [1, 2, 34, 65, 67, 76, 85]. However, some studies including REFL [2] assume a fixed time window for availability [34, 65], which is not always realistic. The availability is influenced by factors such as network status, energy levels, computational capabilities, and user activity. Furthermore, while many studies focus on accuracy, they often don't consider resource inefficiencies caused by client dropouts [2, 8, 39, 51].

4 Motivation

In this section, we motivate our work by examining the limitations of prior research and the factors impacting FL performance.

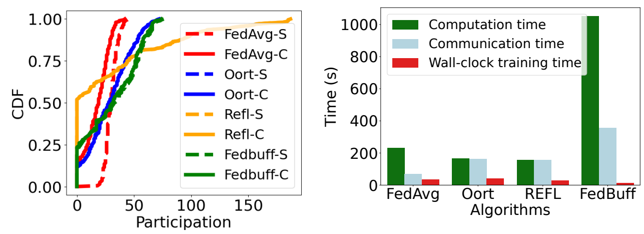
4.1 Limitations of prior art.

Numerous recent studies have employed intelligent client selection strategies to account for heterogeneity in FL [25, 39, 65, 72], which, despite their advancements, have demonstrated declining performance as sources of heterogeneity increases. Other research works, such as AutoFL [34] and FedMarl [84], have adopted reinforcement learning (RL) only for the purposes of having an adaptive and intelligent client selection method. Oort [39] utilizes accuracy and heuristic-based rules for client selection preferring efficient clients which causes bias in selection when a portion of clients have low resources. This bias problem can also be observed with FedBuff which does up to 5× over-selection of clients and selects faster clients more often than clients with limited resources. In addition, it is the least resource-efficient among all client selection algorithms. While some research works such as REFL [2] consider predicting the availability pattern of clients and consider availability as a fixed linear window. This is an unrealistic assumption considering that client availability depends on a multitude of factors such as energy consumption, computational capacity, network availability, and network throughput of devices [40].

We demonstrate these issues through an experiment with 200 clients (20 selected per round for 300 rounds) using the non-EMNIST dataset [14], and a Dirichlet distribution [26, 45] with an alpha value of 0.05. For resource heterogeneity, we employ real-world 4G and 5G network traces [50] to capture clients' resource variability over time, along with a real-world compute trace [2]. FedScale [38] is used for computing latency and other resource metrics, following the practices outlined in [2, 25, 39]. In addition, we assume no interference from co-located applications which means all resources are dedicated for FL training.

In this experiment, we examine three prominent synchronous Federated Learning (FL) methods, namely FedAvg [49], Oort [39], and REFL [2], alongside a state-of-the-art asynchronous FL technique, FedBuff [51]. Figure 2a illustrates the selection bias of both synchronous and asynchronous client selection algorithms, highlighting all selected clients (C) and clients that successfully participated without dropout (S). The findings indicate that FedAvg does relatively unbiased client selection similar to Oort. On the other hand, REFL [2] shows the greatest bias in client selection, excluding 50% of clients from participation as it prefers faster clients. Likewise, FedBuff exhibits bias in client selection, excluding 25% of clients from ever being chosen due to its preference for consistently selecting and aggregating results from faster clients. However, a notable distinction between synchronous and asynchronous methods in terms of resource consumption is depicted in Figure 2b, revealing that the asynchronous FL method requires less than a third to half the training time of synchronous methods; however, asynchronous FL method has 4.5x to 7x greater resource consumption. Thus, Fedbuff as an asynchronous FL approach focuses on quick learning by using client resources extensively, while FedAvg, REFL, and Oort prioritize resource conservation, leading to longer learning times. These outcomes highlight the necessity for a system heterogeneity-aware solution that minimizes selection bias while enhancing training efficiency and resource utilization.

Several research initiatives have explored training unique models for distinct client groups [4, 18, 48]. Auxo [46] proposes scalable client clustering, improving model performance. However, it does not adequately consider the tradeoff between performance and resource usage. Asynchronous communication between clients and the central server allows slower clients to continue local training based on stale models and to contribute to aggregation when ready [35, 51, 71]. However, asynchronous FL methods can substantially drop training accuracy, introduce bias, and suffer from slow convergence [85]. Similarly, semi asynchronous approaches [44, 62, 79] primarily aim to reduce communication in asynchronous FL but overlook other inefficiencies.



(a) Participation count of selected and completed clients.

(b) Accumulated resource usage of all clients and wall-clock FL time.

Figure 2. Limitations of existing frameworks.

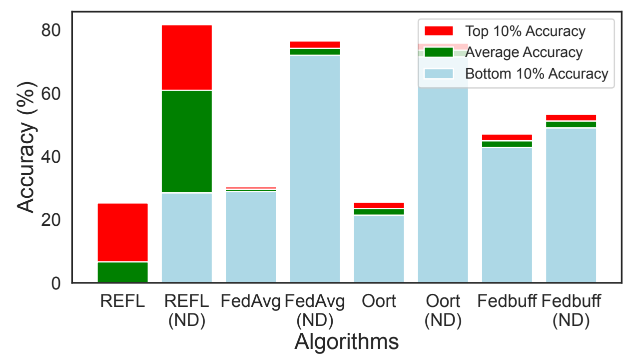


Figure 3. Accuracy of client selection techniques: No dropouts (ND) vs. dynamic interference from co-located apps

4.2 Impact of dropouts on client selection

Despite numerous advances in client scheduling [2, 39, 51], these methods still face challenges in real-world resource conditions. Specifically, dropouts can lead to a considerable decline in accuracy, even when sophisticated client selection strategies are employed. To illustrate this, we employ the same experimental setup as outlined in Section 4.1, and assess the Top 10%, Bottom 10%, and average accuracy metrics of different client selection strategies under two scenarios: assuming no dropouts (ND) and with dropouts (D) under practical resource constraints. The Top 10% accuracy metric averages the accuracy of the highest-performing 10% of clients, while the Bottom 10% accuracy metric does the same for the lowest-performing 10%, and the average accuracy reflects the mean accuracy across all clients. According to the outcomes depicted in Figure 3, all client selection methods experience a significant drop in accuracy due to dropouts, with REFL suffering the most. The primary reason for its vulnerability is its dependence on predicting future client availability, a task it struggles with due to the dynamic nature of client device resources. On the other hand, FedBuff

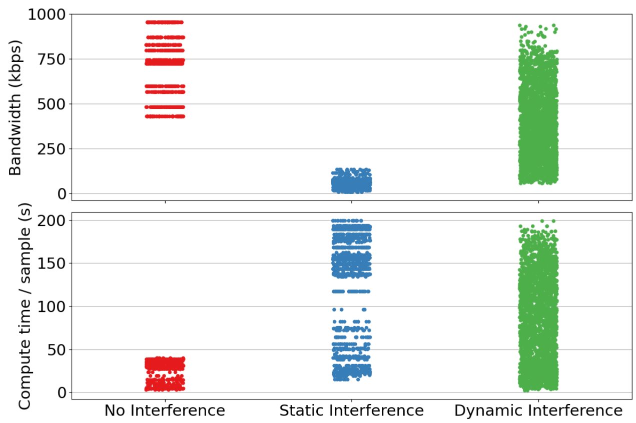


Figure 4. Compute and communication resource variations.

is more resilient to these challenges. As an asynchronous FL algorithm, FedBuff concurrently trains up to 5× more clients, which provides a buffer against accuracy losses. However, this comes at the expense of resource efficiency. In short, the central issue facing these scheduling algorithms is their inability to account for the fluctuating resource conditions at the client's end.

4.3 Limitations of static optimizations

To resolve the challenge of dropouts, various straggler optimizations [29, 57, 61, 66, 83] exist to tradeoff between resource usage, training time, and model performance.

Lossless compression reduces communication bandwidth requirements but needs more computation for compression and decompression. On the other hand, lossy compression and quantization require more computation and can degrade the accuracy of the model. Model pruning saves on computation and communication while partial training decreases computation at the cost of accuracy performance. Each technique neither offers the same reduction in resource utilization nor has a consistent impact on accuracy performance degradation. Furthermore, using all techniques simultaneously isn't feasible due to their combined overheads. It's crucial to assess these methods to understand their resource demands and performance relative to each other. Questions arise, such as *how much pruning versus quantization shortens training for a given performance or what configuration of partial training should be chosen to get the best trade-off between accuracy performance and dropout reduction.* These findings are foundational for efficient, resource-aware FL. Clients should leverage these insights to dynamically select the best technique, adjusting as resources and data distribution evolve. Solely relying on a single static method can be inefficient due to its inability to adjust to changing resources.

Using the same conditions and trace data from Section 4 shown in Figure 2, we highlight the issue by evaluating three resource scenarios: 1) **No Interference** - all client resources are fully available for FL training; 2) **Static On-device Interference** - high-priority applications consistently use some client resources; 3) **Dynamic On-device Interference** scenario emulates a dynamic environment where concurrent applications on the client's device dynamically utilize available resources, resulting in varying levels of resource availability. We divide clients into two groups: Successful Clients, who participated after applying the optimization which otherwise would have dropped out, and Dropped Clients, who still faced dropouts post-optimization.

Figure 4 shows the resource distribution across various scenarios. Without interference, there's ample bandwidth, leading to quick on-device training. With static interference, bandwidth reduces, and computational resources are divided, depending on the compute resources reserved for high-priority applications. Dynamic interference covers all possibilities and reflects realistic, variable resource availability, so we focus on this in further evaluations due to its real-world relevance [30, 34, 76].

Figure 5 displays our evaluation of accuracy performance and client participation metrics using static optimization techniques. Notably, different optimizations respond differently to resource changes. While pruning is preferable in the first two scenarios, partial training outperforms under dynamic interference. Even within a single optimization technique, outcomes differ based on configurations. As demonstrated in Figure 5 (second row), when no interference is present, a mere 25% model pruning is most effective due to the complete allocation of resources to FL training, resulting in decreased dropouts. However, under static interference, resources are consistently used by other priority apps, requiring 75% pruning to ensure client participation. For dynamic interference, 50% pruning provides the best balance, as 25% leads to more dropouts and 75% reduces accuracy. The resource scenarios are not just limited to these three examples. Numerous resource scenarios and configuration settings exist, leading to a vast optimization search space. Hence, static methods are not suitable as a holistic solution for managing the balance between client participation and model performance.

4.4 Heuristics-based solution

Our observations from Figure 5 demonstrate that independent techniques yield inconsistent results in different situations, frequently leading to sub-optimal outcomes. Additionally, no existing solution effectively manages optimizations. Therefore, we propose a heuristic-based approach as a possible solution to address the aforementioned challenges. In this approach we derive the following rules: (1) When the CPU and Network availability for FL training is low (i.e. S_CPU and $S\_Network < Moderate$) due to corunning applications or other multitude of factors, we use more extreme optimization: 75% of either pruning or partial training or 8-bit quantization. (2) If the client devices have sufficient CPU and Network resources (i.e. S_CPU and

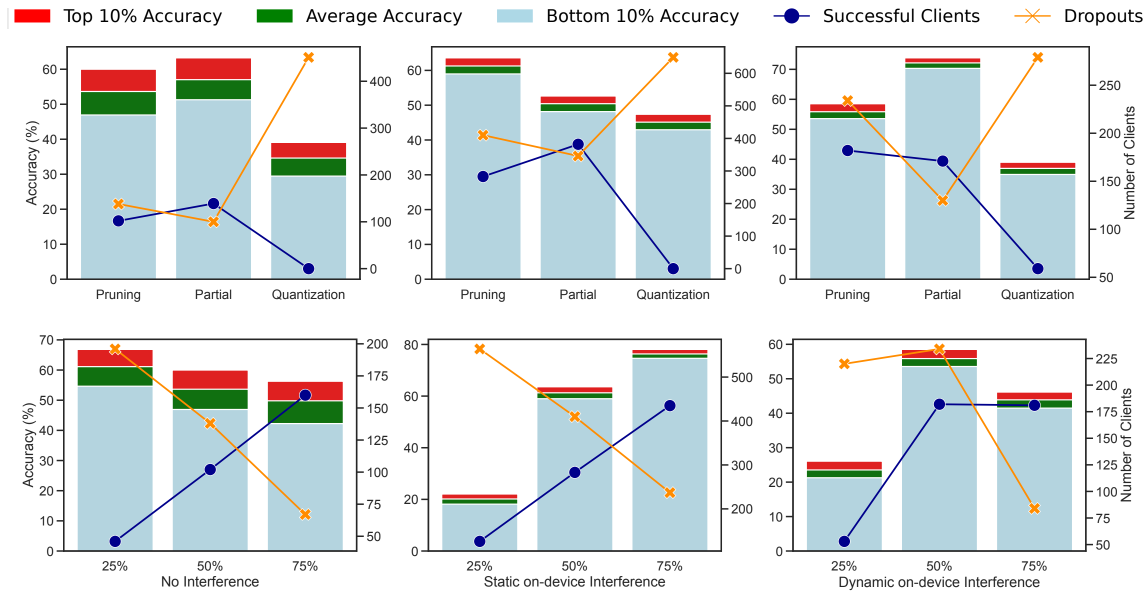


Figure 5. Accuracy, successful and dropped clients of static (top) vs static pruning (bottom) optimizations.

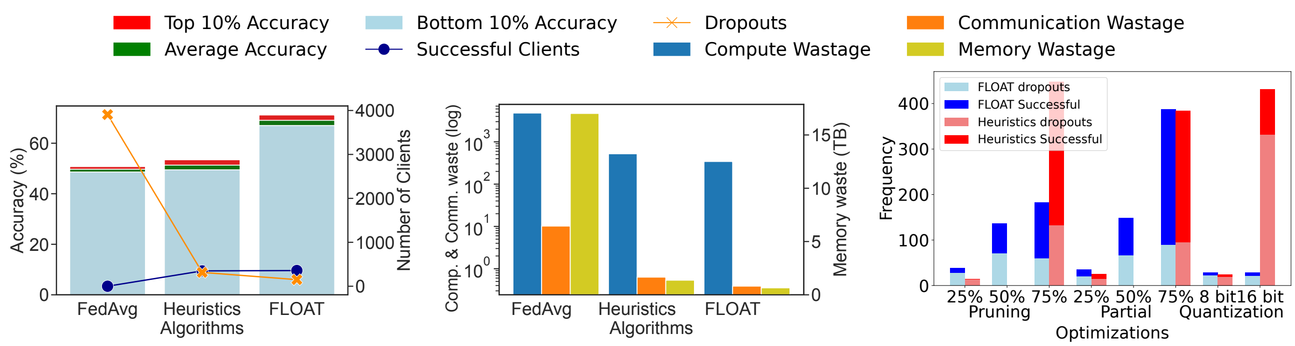


Figure 6. Heuristics vs. FLOAT. Accuracy, Successful and Dropped Clients. (left) Compute, Communication, and Memory inefficiency. (mid) success and failure count of each optimization technique on log scale (right) with the FEMNIST Dataset.

S_Network >= Moderate), we use less extreme optimizations: 16-bit quantization or 25% partial training or pruning. Though optimization selection is random, its configuration is chosen intelligently through the rules above.

We provide a comparison of this heuristics-based approach along with FLOAT with FedAvg as the baseline client selection algorithm. In this experiment, we use a non-IID FEMNIST dataset (Dirichlet alpha 0.01) with dynamic on-device interference. This means that the resources of clients are not only restricted by other high-priority co-located applications, but their requirements also change dynamically.

From Figure 6, we see that using only heuristics for optimization in fluctuating resource conditions isn't optimal. Specifically, Figure 6 (left) reveals the heuristics-based solution surpasses vanilla FedAvg in accuracy and client participation, yet FLOAT enhances accuracy further by nearly 20%. This improvement from FLOAT, as shown in Figure 6 (center), stems from its reduced client dropouts and efficient resource use. To further understand the reduction in dropouts with FLOAT, we assessed the selection patterns of various optimization methods and their configurations. We also examined their respective success rates under both FLOAT and the heuristic approach, which is detailed in Figure 6 (right). FLOAT consistently outperforms, indicating its adeptness in picking the best optimization and configuration.

The heuristic method tends to prefer Quantization, 75% pruning, and partial training. It is worth noting that Quantization's effectiveness diminishes under limited resource scenarios and seems more ideal when the network is the primary constraint, a pattern corroborated by Figure 10. This heuristic bias led to more dropouts, resource wastage, and decreased accuracy, highlighting the shortcomings of solely heuristic-based strategies, especially when faced with complex choices and changing scenarios. The previous motivational points and supporting results raise the following key research questions which we try to address in this work.

Research Questions

RQ1. How to dynamically choose the best technique to tradeoff between model performance, training time, and resource usage and, more importantly, configure it properly?
RQ2. How to manage the overhead of RLHF training at scale?
RQ3. How to adapt the solution to new workloads?
RQ4. How to embed human feedback in RL?
RQ5. How to make the solution scalable for possibly unlimited system conditions?
RQ6. How to define the rewards and a balanced exploration policy of the RLHF agent in FLOAT?
RQ7. How to ensure continuous feedback from dropout clients?

5 FLOAT

In contrast to existing methods that predict each client's behavior for client selection [2, 34, 67], FLOAT actively harnesses clients' resource strengths with feedback integration. This empowers them to join the global update without experiencing staleness. FLOAT achieves this by accelerating straggling clients and optimizing resource and model performance tradeoffs, going beyond basic parameter tuning.

RQ1: Automated Tuning. The major obstacle in designing a holistic solution to provide various available tradeoffs is that the optimization space is too large due to the scale of decentralized training, system and data heterogeneity, runtime variance [34], and various types of possible tradeoffs. To this end, FLOAT uses an automated adaptive prediction mechanism based on RL with Human Feedback (RLHF) [20]. Figure 7 shows the overall design of our approach. The goal is to train an RLHF agent to generate a per-client lookup table. The RLHF agent takes global and client states as input to select optimization actions that achieve a tradeoff strategy for client devices, maximizing the resource efficiency of FL while satisfying model performance requirements. *Global states* from the aggregator include the global model architecture and its parameters; *local states* from clients include compute, network, memory, energy, and model update characteristics; and *optimization actions* include tradeoff techniques such as compression, quantization, pruning, and their hyperparameters. FLOAT uses a Q-learning-based multi-objective RL, where each client's Q-table stores its states (global, local), actions, and Q values. Global states include global model architecture, whereas local states include clients' resource information such as compute, network, and energy capacity. Action space includes the optimization techniques to select and configure. The Q-value ($Q(s, a)$) represents the expected cumulative reward an agent can achieve by taking an action 'a' in a state 's' and following an optimal policy from that point forward. The Q-values in Q-learning, traditionally updated using Bellman's Equation 1, involve parameters: α (learning rate), $R(s, a)$ (immediate reward for action 'a' in state 's'), and γ (discount factor indicating the value of future versus immediate rewards). $\max_{a'} Q(s', a')$ denotes the maximum Q-value for potential actions in the next state 's'. *The key concept is that the Bellman equation relates the Q-value of a state-action pair to the Q-values of subsequent states and actions.* The Bellman's equation is defined as follows:

$$Q(s,a) = (1-\alpha)\cdot Q(s,a) + \alpha \cdot [R(s,a) + \gamma \cdot \max_{a'} Q(s',a')] \quad (1)$$

$Q(s', a')$, representing the new state, is independent of the previous action taken by the RLHF agent. Instead, it is dependent on the dynamic resource availability of the client which is unpredictable and random. So *we update Bellman's equation of Q-learning to reduce the term* $\lim_{\gamma\to 0} \gamma$ *such that the updated* Q *value is not influenced by the random new state.* The reward function, aligned with FLOAT's objectives of enhancing model accuracy and improving client participation, is defined by Equation 2. Here, P_i represents client i's participation success, and Acc_i denotes its accuracy improvement. w_p and w_a assign weights to each objective.

$$R_i = w_p \cdot P_i + w_a \cdot Acc_i \quad (2)$$

Including accuracy in the multi-objective reward function allows FLOAT to manage the non-linear impact of different acceleration configurations on accuracy. Furthermore, by utilizing RLHF for probabilistic exploration and exploitation in each round, FLOAT can automate the straggler accelerator to enhance performance, reduce bias, and improve resource efficiency by proactively minimizing dropouts. *A significant benefit of our approach is its adaptability, as it is not limited to synchronous FL. We have also integrated FLOAT with asynchronous FL and FLOAT can also be used with non-horizontal FL as discussed further in Section 7.*

RQ2: Overhead of RLHF at Scale. Clients comfortable with sharing system usage data with a central aggregator, as seen in standard FL protocols [34], can centralize the per-device lookup table training. However, for privacy-conscious clients, this training can occur locally, incurring a minor additional training cost of under a millisecond, encompassing communication cost [34, 80]. Thus, FLOAT can be scaled to any number of clients with no additional overhead. Figure 8 highlights the memory overhead associated with training the RLHF agent, charting memory use against an escalating count of states and actions. Given FLOAT's potential state and action combinations, the **memory overhead is under 0.2 MB**, and the RLHF agent's **training overhead is**

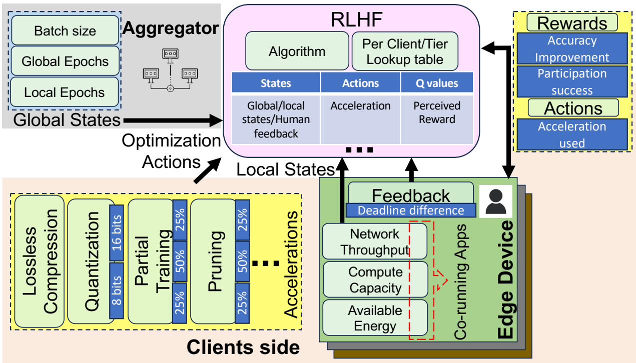


Figure 7. FLOAT architecture

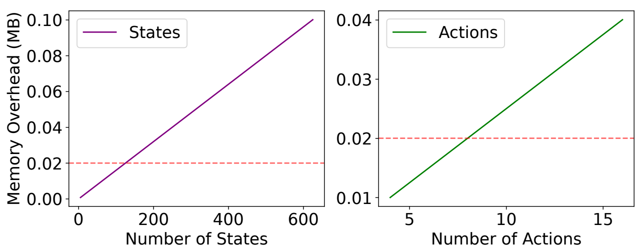


Figure 8. RLHF agent Overhead as the number of states increase, FLOAT uses RLHF agent with 8 actions and 125 possible state combinations (shown by red horizontal line)

less than one millisecond for each training round. This includes the total time taken from choosing the actions and updating the Q-table. This overhead is minimal compared to the lengthy aggregation process [32].

RQ3. Fine-tuning RLHF for new workloads. FLOAT can *train a collective lookup table at the aggregator* to scale to millions of devices or *utilize a pre-trained RLHF agent*, by efficiently fine-tuning it at minimal cost [54, 89]. Contrary to existing RL-based works [34] addressing heterogeneity, FLOAT only shares system-level resource availability information with the RLHF agent, safeguarding clients' data privacy.

To exemplify, we initially pre-train the RLHF agent in FLOAT using the ResNet-18 model, adhering to the configurations outlined in Figure 5 within section 1, with the FEMNIST dataset. The RLHF agent achieves convergence after around 200 rounds of training. Subsequently, we transfer this pre-trained RLHF agent to another benchmark dataset, namely FL with the CIFAR10 dataset. Figure 9 presents the average reward obtained by the RLHF agent, factoring in both model performance and participation success as objectives. These findings attest to the RLHF agent's ability to fine-tune effectively, quickly converging when applied to the new CIFAR10 dataset. To further demonstrate the RLHF agent's adaptability across various FL scenarios, we employ the RLHF agent pre-trained on the FEMNIST dataset and ResNet-18 model

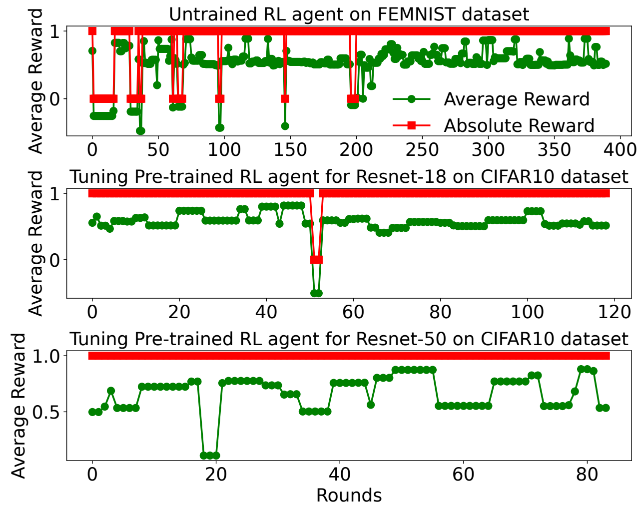


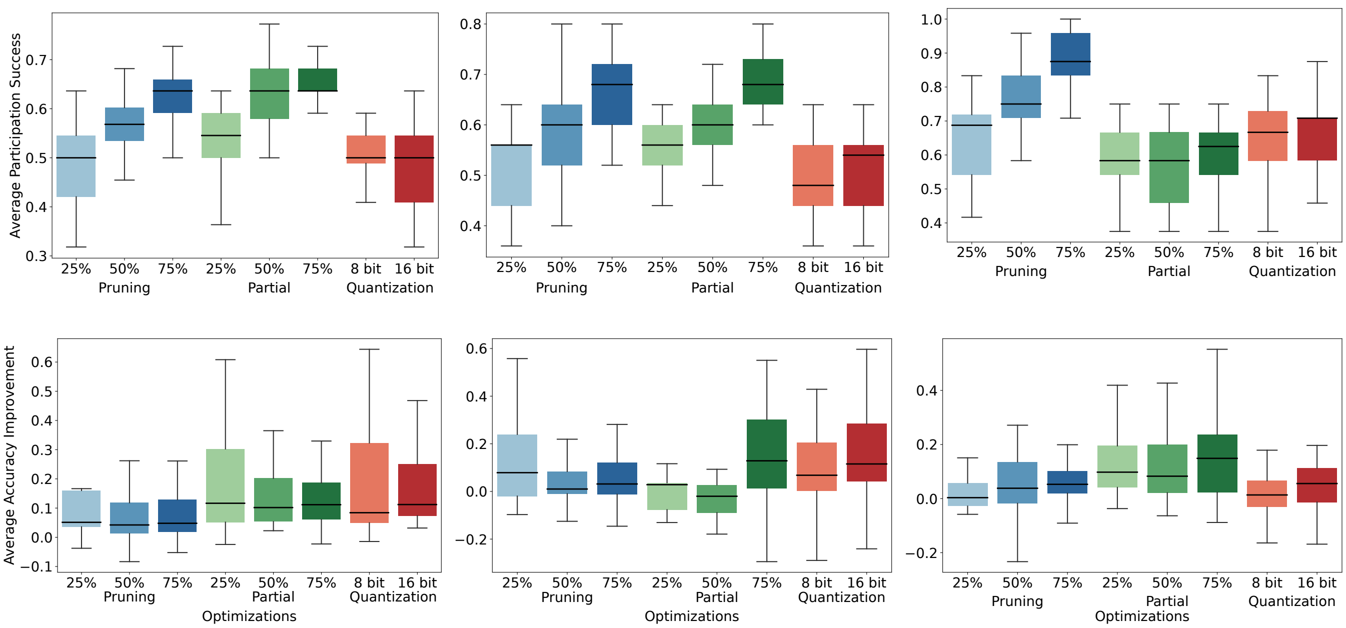
Figure 9. Reusability of RLHF agent

and deploy it in FL training on the CIFAR10 dataset with the ResNet50 model. Remarkably, the RLHF agent undergoes fine-tuning in just 20 rounds of training, yielding positive rewards measured by absolute rewards. The choice of the absolute reward metric is deliberate, as the average reward rarely reaches 100% due to its dependence on accuracy improvement, which realistically remains below 100%. *These results underscore the feasibility of initializing FL with a pre-trained RLHF agent and fine-tuning it for the local context within a few rounds, without incurring significant training overhead.*

Figure 10 represents the use of a pre-trained RL agent, fine-tuned for three unique resource scenarios. The figure shows the multi-objective Q-table of the RLHF agent adjusted for each scenario. From the results, we discern differences in participation success and accuracy enhancement when applying various acceleration techniques and configurations across different resource contexts. This highlights that compared to the limitations of static optimizations and heuristic solutions in adapting to new resource conditions, FLOAT seamlessly adapts via online learning with its RLHF agent.

In particular, Figure 10a reveals that when the data is IID, accuracy improvement remains relatively stable. This stability can be attributed to the fact that with IID data, dropouts do not significantly compromise accuracy, as clients dropping out possess similar data distributions to participating clients, resulting in minimal information loss during training. There is only a slight degradation in accuracy, however, as we move from applying less aggressive optimization (25% pruning or partial training and 16-bit quantization) to more aggressive optimization (75%, 8-bit). We see this same trend repeated in participation success. As we transition from less aggressive to more aggressive optimization methods, the participation success rate intuitively increases.

Additionally, we observe that in the case of an unstable network, as depicted in Figure 10c, partial training exhibits the



(a) IID Dataset. **(b)** non-IID Dataset. **(c)** Unstable Network (non-IID data).

Figure 10. Q table actions participation success rate. (top) accuracy improvement. (bottom)

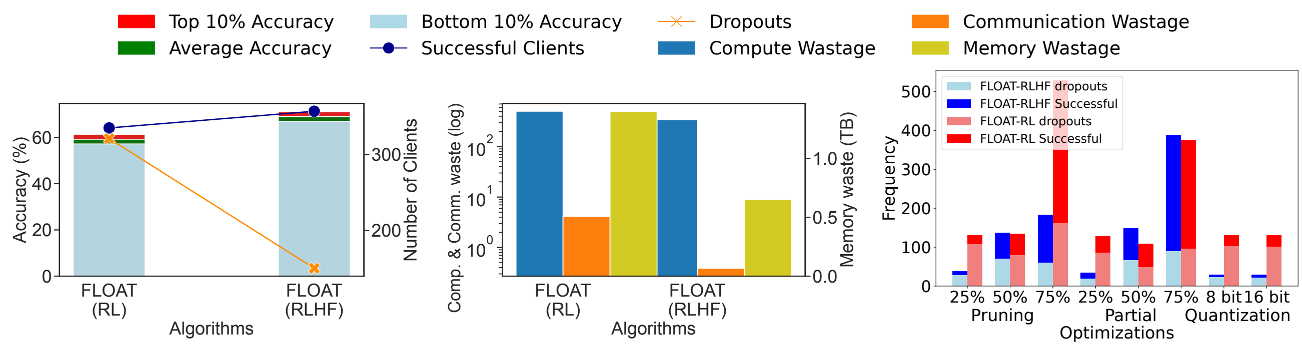


Figure 11. RLHF ablation study. Accuracy, Successful and Dropped Clients. (left) Compute, Communication, and Memory inefficiency. (mid) success and failure count of each optimization technique on log scale (right) with the FEMNIST Dataset.

lowest success rate, while Quantization and Pruning outperform it in terms of participation success. This divergence can be attributed to the fact that partial training primarily alleviates the computational burden, whereas Pruning and Quantization also alleviate the communication burden. Consequently, when communication becomes a bottleneck, partial training underperforms, a lesson learned and depicted in the Q-table of the fine-tuned RLHF agent in FLOAT.

RQ4: Embedding human-feedback. FLOAT enhances the coordination between clients and aggregators by incorporating human feedback, given its intrinsic link to device behavior and resource usage patterns [2, 28, 50]. In the context of Reinforcement Learning (RL), human feedback refers to the insights or information provided by human users, which aims to guide and refine the training and decision-making processes of RL algorithms [13, 20]. This type of feedback is invaluable. Within Reinforcement Learning (RL), human feedback supplies information from users to refine RL algorithm training and decisions [13, 20]. Such feedback is critical for decisions regarding client participation, hinting when a client might rejoin or leave a session. Furthermore, it supports the RLHF agent in aligning with a client's unique resource profile, simplifying agent tuning and improving the convergence speed of both the RLHF agent and the overall FL process. In addition to a holistic understanding of resources, FLOAT extracts more specific, fine-grained details through human feedback. It specifically utilizes feedback concerning deadline differences, along with additional system resources information obtained from clients. This deadline difference

Table 1. Global Parameters, Runtime Variance, and Human Feedback

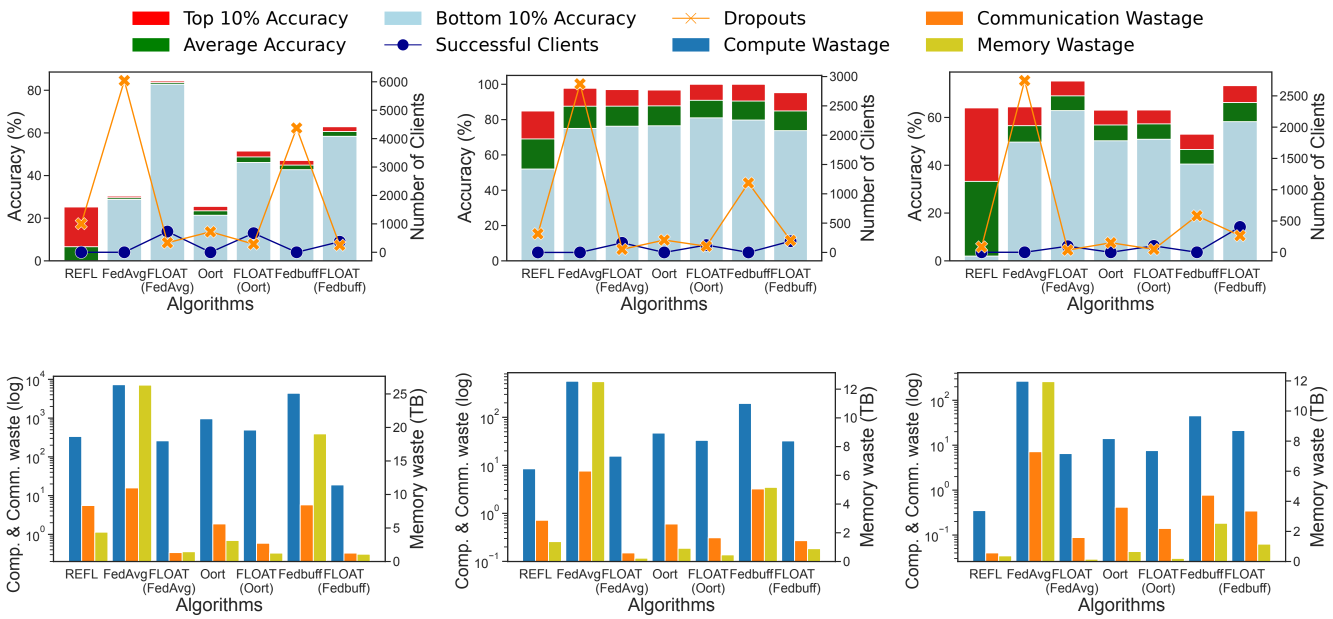
Global Parameters	Description	Discrete Values
G_B	Batch size	Small (< 8), medium (8 − 31), large (≥ 32)
G_E	# of local epochs	Small (< 5), medium (5 − 9), large (≥ 10)
G_K	# of participant devices	Small (< 10), medium (10 − 49), large (≥ 50)
Runtime Variance		
S_{CPU}	Available CPU resources	None (0%), Low (1-20%), Moderate (21-40%), High (41-60%), Very High (61-80%)
S_{MEM}	Available Memory resources	None (0%), Low (1-20%), Moderate (21-40%), High (41-60%), Very High (61-80%)
S_{Network}	Available Network resources	Low (1-20%), Moderate (21-40%), High (41-60%), Very High (61-80%), Extremely High (81-100%)
Human Feedback		
Deadline difference	Client's missed deadline (percentage more time than set time)	None (0%), Low (< 10%), Moderate (< 20%), High (< 30%), Very High (>= 30%)

shows how much a client typically deviates from the prescribed training round deadline.

To gauge the benefits of embedding human feedback (HF) into FLOAT's RL agent, we conducted an ablation study comparing an RL agent with HF (FLOAT-RLHF) to one without (FLOAT-RL). In this evaluation, we used the dynamic on-device interference setting, characterized by client resource conditions that are in a state of flux. The insights derived from this study, as depicted in Figure 11, unveil that the RL agent, when combined with HF, distinctly outperforms the standard RL agent. The results indicate a 10% accuracy boost and 2× fewer client dropouts in Figure 11 (left). Resource utilization of communication, computation, and memory improves by 1.5×, 10×, and 2× in Figure 11 (center). The improvement is largely due to RLHF using human feedback. It employs this feedback to fine-tune its decision-making mechanism and make informed updates to the Q table. We studied the selection patterns, success, and dropout rates of optimization methods in both FLOAT-RLHF and FLOAT-RL. Figure 11 (right) shows that without human feedback fine-tuning, FLOAT-RL is less effective, resulting in poor success-to-dropout ratio than FLOAT-RLHF. FLOAT-RL over-selects clients for optimization, favoring 16-bit quantization and 75% pruning, resulting in more dropouts and reduced accuracy and resource efficiency. In essence, human feedback equips the RLHF agent with vital insights, enhancing its ability to manage resource dynamics via online learning.

RQ5: Scaling RLHF via dimensionality reduction. Both system resources at clients and the human feedback, particularly the deadline difference, are characterized by continuous values. This poses a unique challenge, especially since Q-learning with RLHF employs discrete values for the Q-table. Such continuous metrics could lead to an overwhelming array of state possibilities, making the Q-table impractically large. Employing a histogram method to reduce the dimensionality of continuous variables is prevalent, transforming continuous metrics into discrete bins. However, the choice of bin count is pivotal; it directly influences the granularity of data retained during the continuous-to-discrete conversion. To ensure we capture an optimal amount of information from both system resources and human feedback, we adopted a statistical dimensionality reduction approach. This method begins by determining the variance of resource metrics (computation, communication, energy, deadline difference) at clients. Subsequently, using this variance, we establish percentile boundaries for the bins. After extensive evaluation, we discerned that designating 5 discrete states for these metrics offers the most balanced performance. Fewer than 5 states compromise the richness of information and decelerate the convergence speed of the RL agent paired with Q-learning. Conversely, exceeding 5 states magnifies exploration time for marginal performance gains. The global parameters, runtime variance, and human feedback variables that increase the search space are shown in Table 1. Adding new acceleration techniques increases the number of actions, thereby expanding the action space. However, unlike deep reinforcement learning, the Q-learning approach used in FLOAT is simpler and more efficient. The search space in Q-learning is defined by the number of states ($\mathcal{S}$) and actions ($\mathcal{A}$). Adding a new acceleration technique increases the actions by one, thus expanding the exploration space of the RL agent by $\mathcal{S}$. With the statistical dimensionality reduction of $\mathcal{S}$, FLOAT ensures a linear and minimal increase in the search space, which aids in efficient convergence and fine-tuning of the RL agent at scale.

RQ6: Rewards and exploration policy of RLHF. Reward selection plays a pivotal role in determining the RL agent's convergence speed. Initially, we observed that taking accuracy directly as a reward caused two issues. First, actions chosen often during exploration had higher total accuracy reward scores due to the additive nature of Bellman's Equation 1. To resolve this, we shifted from direct score accumulation to computing its moving average. This was also



(a) FEMNIST Dataset. **(b)** Speech Dataset. **(c)** CIFAR10 Dataset.

Figure 12. Accuracy, Successful and Dropped Clients. (top) Compute, Communication, and Memory inefficiency. (bottom)

replicated for the other reward, namely participation success, ensuring consistent scores across each objective in our multi-objective RLHF agent. Second, we recognized a temporal variation of accuracy across rounds, complicating its inclusion as a consistent reward during RLHF agent updates. Notably, the growth in accuracy during initial rounds is significantly more than in later rounds. To resolve this, we introduced a dynamic learning rate for RLHF training. This rate is controlled by the progress in the FL process. While the learning rate starts off low during the early rounds, it gradually escalates throughout training but never exceeds a cap of 1.0. Furthermore, we identified an imbalance in action selection, favoring specific acceleration configurations. To counteract this, we adjusted our exploration function to prioritize lesser-explored actions. This deliberate alteration ensured a more balanced exploration across all actions resulting in an easily tuneable RLHF agent.

RQ7: Receiving feedback from dropout clients. Training an RLHF agent requires feedback to learn after each action. However, inferring for every client after each cycle is computationally intensive. So, we prioritize validation for active contributors, especially those that experienced acceleration. This feedback updates the RLHF agent's Q-table. A challenge arises when clients that applied the acceleration technique dropout, causing the RLHF training to miss their feedback. To address this, we cache feedback from similar clients and combine it with the dropped client's past improvements to estimate their rewards. This ensures the RLHF gets feedback for all actions resulting in faster convergence. We provide the steps of FLOAT's design in Algorithm 1.

6 Evaluation

6.1 Evaluation Setup

For the end-to-end, we use NVIDIA GeForce RTX 3070 GPUs to compare FLOAT with other algorithms across various models and datasets. Our FLOAT implementation is built on FedScale [38], a platform recently used as a base for other significant contributions in this field [2, 16, 39].

System configuration: We use three realistic traces in our simulation environment to mimic client availability, computational capacity, and network bandwidth, mirroring true real-world resource conditions.

Client Availability Trace: We simulate the energy availability conditions for client devices using an availability trace from [76]. This trace offers insights into client availability for training based on their residual energy levels.

Compute Availability Trace: Drawing from [27], we utilize a compute trace that details the training time across over 950 diverse mobile and edge devices for 25 distinct models.

Network Bandwidth Trace: To reflect real-world network conditions, we incorporate a network trace from [50]. This trace includes data from 4G and 5G network environments recorded on various mobile devices under assorted conditions, such as during vehicular movement or while walking.

Specifically, our simulations employ the dynamic on-device interference system configurations as depicted earlier in Figure 4, ensuring comprehensive coverage of potential system resource scenarios. *To the best of our knowledge, our simulator stands unparalleled in terms of realism for FL workloads.* Oort uses compute and network data to simulate client response

Algorithm 1 Training the Q-Learning Model

Variable: S_{global}, S_{local}, A
S_{global} is the global state
S_{local} is the local state
A is the action (execution target)
Constants: $r_c, r_t, \gamma, \mu, \epsilon$
r_i is the current training round
r_t is the Total training rounds
γ is the learning rate $\leftarrow$ max $(\frac{1.0*r_i}{r_t}, 1.0)$
μ is the discount factor
ϵ is the exploration factor
Intialize Q(S_{global}, S_{local}, A) as random values
Repeat (whenever an aggregation round begins):
Observe global state and store in S_{global}
Observe local state for each device and store in S_{local}
if rand()< ϵ **then**
Choose K participants (based on employed client selection algorithm)
Choose action A randomly for selected participant
else
Sort optimizations by Q($Sglobal, S_{local}, A$)
Choose action A with the largest Q(S_{global}, S_{local}, A)
Run training on a target defined by A in each device
(when local training and aggregation ends)
Get P_i, Acc_i and calculate reward R_i via Equation 2
Observe new global state S'_{global}
Observe new local state S'_{local}
Sort devices by Q($S'_{global}, S'_{local}, A'$)
Choose action A' with the largest Q(S'_{global}, S'_{local}, A')
$Q(S_{global}, S_{local}, A) \leftarrow Q(S_{global}, S_{local}, A)$
$+ \gamma[R_i + \mu\, Q(S'_{global}, S'_{local}, A')$
$- Q(S_{global}, S_{local}, A)]$
$S \leftarrow S'$

times for selection, but doesn't directly assess resource abundance or scarcity. REFL similarly relies on response times without distinguishing resource surplus or shortfall, key for informed optimization and configuration choices. REFL stands out by predicting future windows but simplifies client availability as a one-dimensional window. In addition to this, both Oort and REFL consider that resources such as computation and network capacity will always remain constant at clients, which is another unrealistic assumption [15, 50]. AutoFL [34] exclusively focuses on energy utilization. In synchronous FL, seen in works like [49] and FedBuff [51], dynamic client resource availabilities are generally overlooked. None of these approaches consider the combined dynamics of compute, network, energy, and memory resources, vital for adaptive optimization and configuration selection.

Datasets, Models, and Tasks: We test on three distinct datasets: CIFAR10 [36], FEMNIST [7], and OpenImage [37] for vision tasks, and a Google dataset for speech recognition [69] with Dirichlet alpha 0.1, ensuring varied evaluation of our method. Client selection studies [2, 34, 39, 49, 51] often assume the aggregator server has holdout IID datasets. This is impractical as a true IID dataset requires knowing clients' private data distribution, inaccessible by aggregator. Hence, we rely on calculating accuracy using the non-IID dataset at clients. To standardize, we use consistent configurations in all tests. We use the Resnet34 model, with a learning rate of 0.01, a batch size of 20, and 5 local epochs per training round, totaling 300 rounds. From a global client pool of 200, we sample 30 clients per training iteration. In the FedBuff context, we let 100 clients train simultaneously and asynchronously, keeping a buffer of 30 clients.

To our knowledge, limited research has addressed adaptive optimization selection and configuration. Hence, we compared FLOAT with four notable client selection algorithms: REFL [2], Oort [39], and FedAvg [49], which are synchronous FL algorithms, and FedBuff [51], an asynchronous FL algorithm. Importantly, we did not use FLOAT with REFL due to fundamental differences in their assumptions. REFL presumes clients' future availability windows are predictable, a premise we find unrealistic considering the many factors affecting client window dynamics, including system, network, energy, and client willingness [15, 30, 50, 53].

Metrics: We then incorporate FLOAT into the baseline methods and evaluate their performance using three metrics. First, we measure the accuracy of the top 10%, average, and bottom 10% of clients to identify potential biases and gauge overall performance. Next, we track the number of dropouts during training and those retained after FLOAT's adaptive optimizations. Lastly, we assess resource inefficiency, looking at total computation and communication time in hours and memory inefficiencies in TB from client dropouts. Together, these metrics provide a thorough assessment of the enhancements brought by FLOAT's integration into the baseline methods.

6.2 End-to-end performance

We conducted an end-to-end evaluation of FLOAT, comparing its performance to standard algorithms as illustrated in Figure 12, spanning three distinct datasets. Our findings revealed significant accuracy enhancements when using FLOAT across all algorithms, with particularly notable improvements with the FEMNIST and CIFAR10 datasets. The most pronounced accuracy improvements were observed with FedAvg and Oort.

FedBuff, an asynchronous FL algorithm, concurrently trains five times more clients than its synchronous counterparts. It employs a buffer for result aggregation, offering resilience to resource fluctuations and client dropouts. Thus, pairing FLOAT with FedBuff does not provide as much improvement as with synchronous FL. However, FedBuff's over-selection policy culminates in notable resource inefficiencies, as depicted in Figures 12a and 12c and FLOAT integrated with FedBuff significantly reduces this inefficiency. FedAvg, while

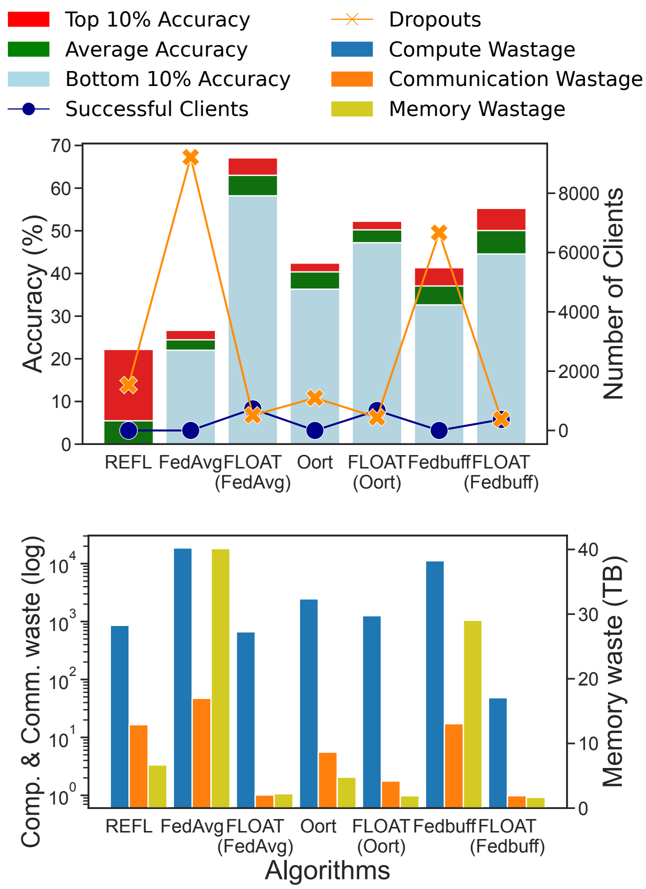


Figure 13. Performance on the OpenImage dataset [37]. Accuracy, Successful and Dropped Clients. (top) Compute, Communication, and Memory inefficiency. (bottom).

bearing similarities to FedBuff in terms of resource inefficiency, adopts a rudimentary client selection method, opting for random client selection without accounting for resource availability. Thus, we see the most prominent improvement with FLOAT (FedAvg). In contrast, REFL exhibits the poorest accuracy performance because it overly relies on predicting client availability windows. Unfortunately, it assumes that client resources will remain constant, resulting in suboptimal predictions in dynamically changing environments, leading to even more dropouts compared to other algorithms.

FLOAT, specifically FLOAT (FedAvg) and FLOAT (FedBuff) stand out by delivering the highest accuracy compared to FLOAT (Oort). This can be attributed to the unbiased client selection approach adopted in FedAvg and FedBuff. In contrast, Oort's selection relies on client efficiency, which can prove sub-optimal in a dynamically changing resource environment. A client that performed efficiently in one round may not exhibit the same efficiency in subsequent rounds.

FLOAT also enhances model accuracy shown in Figure 12 (first row) by 16% to 53% for the FEMNIST dataset, 1% to 20% for CIFAR10, and up to 3% for the Speech dataset. The performance of FLOAT remains consistent in terms of accuracy with the Speech dataset, primarily due to its lower resource requirements for training and faster convergence speed. Consequently, it experiences fewer dropouts, providing limited opportunities for FLOAT to further reduce dropout rates and provide any substantial improvements in accuracy.

Oort, as previously illustrated in Figure 2a, exhibits a bias toward selecting efficient clients. FLOAT effectively reduces the total number of dropouts across 300 training rounds by a remarkable 3×-18× for the FEMNIST dataset, 3×-78× for CIFAR10, and 2×-54× for the Speech dataset. This reduction has a cascading effect on resource inefficiencies. When clients remain active, their progress contributes to the FL training, and the computational and communication time invested in training and transmitting the model to the aggregator becomes a valuable part of the final aggregated model. In contrast, when a client drops out, the energy, communication, computation, and memory resources invested in its training, transportation, and storage are wasted, as its input no longer contributes to FL. Furthermore, the resources allocated for FL training could have been employed by other co-located applications, resulting in an opportunity cost.

We also quantify computation and communication inefficiencies by calculating the time spent on model training and the round-trip communication time for updating the model. Memory inefficiency is calculated as terabytes (TBs) of memory used during training and model storage. The results are displayed in the second row of Figure 12. We calculate these inefficiencies across all 200 clients with 30 clients selected per round for synchronous FL and 100 clients selected for concurrent training in asynchronous FL (FedBuff) over the span of 300 training rounds, each comprising 5 local epochs. FLOAT notably improves resource efficiency, especially with FedAvg and FedBuff. FedBuff over-allocates resources to address client dropouts, resulting in significant inefficiency of resources, while FedAvg's random client selection [49] chooses clients prone to dropout. FLOAT considerably improves the total computation efficiency, saving 468 to 7013 hours ($2\times-28\times$) for FEMNIST, 7 to 259 hours ($2\times-44\times$) for CIFAR10, and 13 to 547 hours ($1.4\times-36\times$) for Speech dataset. Communication time, based on a 4G trace, sees reductions of 0.3 to 7 hours ($3\times-47\times$) for FEMNIST and CIFAR10 ($3\times-81\times$), and 0.3 to 7.4 hours ($2\times-51\times$) for the Speech dataset. Memory efficiency is improved by 2 to 25 TBs ($3\times-14\times$) for FEMNIST, 0.5 to 12 TBs for CIFAR10 ($2\times-12\times$), and 0.7 to 12 TBs for Speech dataset ($2\times-12\times$).

Performance on complex datasets: We further evaluate the performance of FLOAT in comparison to the state-of-the-art on more complex datasets. For this purpose, we use the OpenImage dataset [37] consisting of 1.6 million images. Additionally, we use all the same settings described in Section 6.1. However, in place of the Resnet34 model, we use the

ShuffleNet model [87], aligning with the model choice made in related works assessing the OpenImage dataset [2, 39].

The findings presented in Figure 13 corroborate with those observed in Figure 12. The FedAvg algorithm, lacking a sophisticated mechanism for client selection, tends to choose clients with a higher likelihood of dropout. In contrast, Oort demonstrates improved performance due to its strategy of selecting clients with a greater probability of completing their training. REFL, however, places excessive emphasis on predicting client availability windows, making it the most vulnerable to dropouts. FedBuff shows comparable performance with Oort by over-selection of clients to offset performance degradation from dropouts. However, over-selection results in increased resource inefficiency. FLOAT, on the other hand, significantly enhances both accuracy and resource efficiency, especially with FedAvg and FedBuff. Overall, with the OpenImage dataset, FLOAT improves the accuracy by 8% to 39%. FLOAT also improves the cumulative clients' computation resource efficiency by 1200 to 1160 hours ($3\times - 233\times$), total communication efficiency by 3.8 to 46 hours ($3\times - 46\times$), and total memory efficiency by 3 to 38 TBs ($2\times - 20\times$).

7 Discussion

Revamping the design of client selection algorithms: Another interesting observation as shown in Figure 12, highlights that random selection algorithms counter-intuitively surpass intelligent client selection methods. A closer look reveals that many client selection algorithms operate under the assumption that a client's resources remain consistent throughout training. This idea is foundational in algorithms like Oort [39]. Although recent studies now incorporate predicting fluctuating availability windows of clients [2] or choosing energy-conserving clients [34], they don't fully consider all resource dynamics affecting training performance. These studies typically concentrate on singular factors like set availability windows [2], response speeds [39], or power efficiency [34]. This oversight highlights a significant research gap, underscoring the need to consider the variability of client resources in enhancing client selection strategies.

Limitations of FLOAT: FLOAT excels in cross-device FL with limited resources, optimizing dropout reduction and resource efficiency through adaptive acceleration. However, its benefits diminish in resource-rich environments or when reducing dropouts is less critical, as in FedBuff's over-selection strategy. To enhance accuracy in such scenarios, FLOAT integrates a weighted multi-objective rewards equation into its RLHF agent, allowing users to prioritize accuracy over resource efficiency and dropout reduction.

FLOAT for non-horizontal FL: Vertical Federated Learning (VFL) involves clients with distinct data features using split [59] or top-bottom models [70]. Hybrid FL [86] combines Horizontal and Vertical approaches. Given the uniformity of local client training requirements, FLOAT can easily integrate with both VFL and Hybrid FL without needing structural adjustments.

Table 2. Shortcomings of current FL approaches and their enhancement with FLOAT

Methods	Shortcomings	Enhancement with FLOAT
FedAvg [49] (Synchronous)	Does not consider dynamic client resource availability	Improves resource efficiency by reducing the number of dropouts
Oort [39] (Synchronous)	Assumes constant client resources, leading to suboptimal selections	Enhances client participation through dynamic resource-based acceleration optimization
REFL [2] (Synchronous)	Relies on future availability predictions without dynamic resource considerations, increasing dropouts	Minimizes dropouts by dynamically configuring acceleration for less available clients
FedBuff [51] (Asynchronous)	Over-selection policy results in resource efficiency	Improves resource efficiency and reduces selection bias by accelerating stragglers

8 Conclusion

This paper introduces FLOAT, a multi-objective reinforcement learning system that leverages human feedback to automate the selection and configuration of accelerators in FL training. Our experiments demonstrate that FLOAT is a versatile, efficient, and scalable solution. It can be fine-tuned at a minimal cost for new workloads, resulting in improved accuracy performance and improved resource efficiencies.

9 Acknowledgments

We thank our shepherd, Sara Bouchenak, and our anonymous reviewers for their valuable feedback and suggestions. This work has been partially funded by NSF grants CSR-2106634, CSR-2312785, CCF-1919113, and OAC-2004751, along with UKRI-EPSRC grant EP/X035085/1.

References

[1] Ahmed M. Abdelmoniem, Chen-Yu Ho, Pantelis Papageorgiou, and Marco Canini. 2023. A Comprehensive Empirical Study of Heterogeneity in Federated Learning. *IEEE Internet of Things Journal* 10, 16 (2023), 14071–14083. https://doi.org/10.1109/JIOT.2023.3250275

[2] Ahmed M. Abdelmoniem, Atal Narayan Sahu, Marco Canini, and Suhaib A. Fahmy. 2023. REFL: Resource-Efficient Federated Learning. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 215–232. https://doi.org/10.1145/3552326.3567485

[3] Haider Ali, Dian Chen, Matthew Harrington, Nathaniel Salazar, Mohannad Al Ameedi, Ahmad Faraz Khan, Ali R. Butt, and Jin-Hee Cho. 2023. A Survey on Attacks and Their Countermeasures in Deep Learning: Applications in Deep Neural Networks, Federated, Transfer, and Deep Reinforcement Learning. *IEEE Access* 11 (2023), 120095–120130. https://doi.org/10.1109/ACCESS.2023.3326410

[4] Christopher Briggs, Zhong Fan, and Peter Andras. 2020. Federated learning with hierarchical clustering of local updates to improve training on non-IID data. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–9.

[5] Chris Brook. 2020. Google Fined $57M by Data Protection Watchdog Over GDPR Violations. https://digitalguardian.com/blog/google-fined-57m-data-protection-watchdog-over-gdpr-violations. [Online; accessed 05-June-2022].

[6] David Byrd and Antigoni Polychroniadou. 2020. Differentially private secure multi-party computation for federated learning in financial applications. In *Proceedings of the First ACM International Conference on AI in Finance*. 1–9.

[7] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097* (2018).

[8] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. 2020. TiFL: A Tier-based Federated Learning System. *To appear in ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC)* (2020).

[9] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. 2021. FedAT: a high-performance and communication-efficient federated learning system with asynchronous tiers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 1–16.

[10] Zheng Chai, Hannan Fayyaz, Zeshan Fayyaz, Ali Anwar, Yi Zhou, Nathalie Baracaldo, Heiko Ludwig, and Yue Cheng. 2019. Towards taming the resource and data heterogeneity in federated learning. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)*. 19–21.

[11] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. 2020. Asynchronous Online Federated Learning for Edge Devices with Non-IID Data. In *2020 IEEE International Conference on Big Data (Big Data)*. 15–24. https://doi.org/10.1109/BigData50022.2020.9378161

[12] Gary Cheng, Karan Chadha, and John Duchi. 2021. Fine-tuning is fine in federated learning. *arXiv preprint arXiv:2108.07313* 3 (2021).

[13] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep Reinforcement Learning from Human Preferences. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf

[14] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. 2017. EMNIST: Extending MNIST to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2921–2926.

[15] Swarnava Dey and et al. 2013. Challenges of Using Edge Devices in IoT Computation Grids. In *ICPADS*.

[16] Enmao Diao, Jie Ding, and Vahid Tarokh. 2020. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264* (2020).

[17] FTC. 2019. FTC Imposes $5 Billion Penalty and Sweeping New Privacy Restrictions on Facebook. https://www.ftc.gov/news-events/news/press-releases/2019/07/ftc-imposes-5-billion-penalty-sweeping-new-privacy-restrictions-facebook. [Online; accessed 05-June-2022].

[18] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. 2020. An efficient framework for clustered federated learning. *Advances in Neural Information Processing Systems* 33 (2020), 19586–19597.

[19] Matei Grama, Maria Musat, Luis Muñoz-González, Jonathan Passerat-Palmbach, Daniel Rueckert, and Amir Alansary. 2020. Robust aggregation for adaptive privacy preserving federated learning in healthcare. *arXiv preprint arXiv:2009.08294* (2020).

[20] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. 2013. Policy Shaping: Integrating Human Feedback with Reinforcement Learning. In *Advances in Neural Information Processing Systems*, C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger (Eds.), Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2013/file/e034fb6b66aacc1d48f445ddfb08da98-Paper.pdf

[21] Andrei Hagiu and Julian Wright. 2020. When data creates competitive advantage. *Harvard business review* 98, 1 (2020), 94–101.

[22] Jingoo Han, Ahmad Faraz Khan, Syed Zawad, Ali Anwar, Nathalie Baracaldo Angel, Yi Zhou, Feng Yan, and Ali R. Butt. 2022. Heterogeneity-Aware Adaptive Federated Learning Scheduling. In *2022 IEEE International Conference on Big Data (Big Data)*. 911–920. https://doi.org/10.1109/BigData55660.2022.10020721

[23] Jingoo Han, Ahmad Faraz Khan, Syed Zawad, Ali Anwar, Nathalie Baracaldo Angel, Yi Zhou, Feng Yan, and Ali R. Butt. 2022. TIFF: Tokenized Incentive for Federated Learning. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. 407–416. https://doi.org/10.1109/CLOUD55607.2022.00064

[24] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).

[25] Qiang He, Zeqian Dong, Feifei Chen, Shuiguang Deng, Weifa Liang, and Yun Yang. 2022. Pyramid: Enabling Hierarchical Neural Networks with Edge Computing. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 1860–1870. https://doi.org/10.1145/3485447.3511990

[26] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. 2019. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335* (2019).

[27] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. Van Gool. 2019. AI Benchmark: All About Deep Learning on Smartphones in 2019. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE Computer Society, Los Alamitos, CA, USA, 3617–3635. https://doi.org/10.1109/ICCVW.2019.00447

[28] Lukasz Jedrzejczyk, Blaine A Price, Arosha K Bandara, and Bashar Nuseibeh. 2010. On the impact of real-time feedback on users' behaviour in mobile location-sharing applications. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*. 1–12.

[29] Yuang Jiang, Shiqiang Wang, Víctor Valls, Bong Jun Ko, Wei-Han Lee, Kin K. Leung, and Leandros Tassiulas. 2022. Model Pruning Enables Efficient Federated Learning on Edge Devices. *IEEE Transactions on Neural Networks and Learning Systems* (2022), 1–13. https://doi.org/

10.1109/TNNLS.2022.3166101

[30] Peter Kairouz, H Brendan McMahan, Andrew Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Christina Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and Open Problems in Federated Learning. *Foundations and Trends in Machine Learning* 12, 3-4 (2019), 1–357.

[31] Renuga Kanagavelu, Zengxiang Li, Juniarto Samsudin, Shaista Hussain, Feng Yang, Yechao Yang, Rick Siow Mong Goh, and Mervyn Cheah. 2021. Federated Learning for Advanced Manufacturing Based on Industrial IoT Data Analytics. In *Implementing Industry 4.0*. Springer, 143–176.

[32] Ahmad Faraz Khan, Yuze Li, Xinran Wang, Sabaat Haroon, Haider Ali, Yue Cheng, Ali R. Butt, and Ali Anwar. 2023. Towards cost-effective and resource-aware aggregation at Edge for Federated Learning. In *2023 IEEE International Conference on Big Data (BigData)*. 690–699. https://doi.org/10.1109/BigData59044.2023.10386691

[33] Ahmad Faraz Khan, Xinran Wang, Qi Le, Azal Ahmad Khan, Haider Ali, Jie Ding, Ali Butt, and Ali Anwar. 2023. PI-FL: Personalized and Incentivized Federated Learning. *arXiv preprint arXiv:2304.07514* (2023).

[34] Young Geun Kim and Carole-Jean Wu. 2021. Autofl: Enabling heterogeneity-aware energy efficient federated learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 183–198.

[35] Ram M Kripa, Andy Zou, Ryan Jia, and Kenny Huang. 2023. Papaya: Federated Learning, but Fully Decentralized. arXiv:2303.06189 [cs.LG]

[36] Alex Krizhevsky, Geoffrey Hinton, and Vinod Nair. 2009. Learning Multiple Layers of Features from Tiny Images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf. Technical report, University of Toronto.

[37] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. 2020. The Open Images Dataset V4: Unified Image Classification, Object Detection, and Visual Relationship Detection at Scale. *International Journal of Computer Vision* 128, 7 (March 2020), 1956–1981. https://doi.org/10.1007/s11263-020-01316-z

[38] Fan Lai, Yinwei Dai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. FedScale: Benchmarking model and system performance of federated learning. In *Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning*. 1–3.

[39] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient federated learning via guided participant selection. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 19–35.

[40] 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. https://doi.org/10.1109/MSP.2020.2975749

[41] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems* 2 (2020), 429–450.

[42] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. 2020. Fair Resource Allocation in Federated Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ByexElSYDr

[43] Xiang Li, Lingyun Lu, Wei Ni, Abbas Jamalipour, Dalin Zhang, and Haifeng Du. 2022. Federated Multi-Agent Deep Reinforcement Learning for Resource Allocation of Vehicle-to-Vehicle Communications. *IEEE Transactions on Vehicular Technology* 71, 8 (2022), 8810–8824. https://doi.org/10.1109/TVT.2022.3173057

[44] Feiyuan Liang, Qinglin Yang, Ruiqi Liu, Junbo Wang, Kento Sato, and Jian Guo. 2022. Semi-synchronous federated learning protocol with dynamic aggregation in Internet of Vehicles. *IEEE Transactions on Vehicular Technology* 71, 5 (2022), 4677–4691.

[45] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. 2020. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems* 33 (2020), 2351–2363.

[46] Jiachen Liu, Fan Lai, Yinwei Dai, Aditya Akella, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. Auxo: Heterogeneity-Mitigating Federated Learning via Scalable Client Clustering. *arXiv preprint arXiv:2210.16656* (2022).

[47] Guodong Long, Yue Tan, Jing Jiang, and Chengqi Zhang. 2020. Federated learning for open banking. In *Federated learning*. Springer, 240–254.

[48] Guodong Long, Ming Xie, Tao Shen, Tianyi Zhou, Xianzhi Wang, and Jing Jiang. 2023. Multi-center federated learning: clients clustering for better personalization. *World Wide Web* 26, 1 (2023), 481–500.

[49] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[50] Arvind Narayanan, Eman Ramadan, Jason Carpenter, Qingxu Liu, Yu Liu, Feng Qian, and Zhi-Li Zhang. 2020. A First Look at Commercial 5G Performance on Smartphones. In *Proceedings of The Web Conference 2020* (Taipei, Taiwan) *(WWW '20)*. Association for Computing Machinery, New York, NY, USA, 894–905. https://doi.org/10.1145/3366423.3380169

[51] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Michael G. Rabbat, Mani Malek, and Dzmitry Huba. 2021. Federated Learning with Buffered Asynchronous Aggregation. *CoRR* abs/2106.06639 (2021). arXiv:2106.06639 https://arxiv.org/abs/2106.06639

[52] Jacquelyn K O'herrin, Norman Fost, and Kenneth A Kudsk. 2004. Health Insurance Portability Accountability Act (HIPAA) regulations: effect on medical record research. *Annals of surgery* 239, 6 (2004), 772.

[53] Jianli Pan and James McElhannon. 2018 pages=439-449,. Future Edge Cloud and Edge Computing for Internet of Things Applications. *IEEE IoT-J* (2018 pages=439-449,).

[54] Haoran Qiu, Weichao Mao, Chen Wang, Hubertus Franke, Alaa Youssef, Zbigniew T Kalbarczyk, Tamer Basar, and Ravishankar K Iyer. 2023. AWARE: Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems. In *USENIX Annual Technical Conference*.

[55] Youyang Qu, Shiva Raj Pokhrel, Sahil Garg, Longxiang Gao, and Yong Xiang. 2020. A blockchained federated learning framework for cognitive computing in industry 4.0 networks. *IEEE Transactions on Industrial Informatics* 17, 4 (2020), 2964–2973.

[56] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. 2021. Adaptive Federated Optimization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=LkFG3lB13U5

[57] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. 2020. FedPAQ: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 108)*, Silvia Chiappa and Roberto Calandra (Eds.). PMLR, 2021–2031. https://proceedings.mlr.press/v108/reisizadeh20a.html

[58] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* (1951), 400–407.

[59] Daniele Romanini, Adam James Hall, Pavlos Papadopoulos, Tom Titcombe, Abbas Ismail, Tudor Cebere, Robert Sandmann, Robin Roehm, and Michael A. Hoeh. 2021. PyVertical: A Vertical Federated Learning

Framework for Multi-headed SplitNN. arXiv:2104.00489 [cs.LG]
[60] Adam Satariano. 2021. Facebook's WhatsApp is fined for breaking the E.U.'s data privacy law. https://www.nytimes.com/2021/09/02/business/facebook-whatsapp-privacy-fine.html. [Online; accessed 05-June-2022].
[61] Nir Shlezinger, Mingzhe Chen, Yonina C. Eldar, H. Vincent Poor, and Shuguang Cui. 2021. UVeQFed: Universal Vector Quantization for Federated Learning. *IEEE Transactions on Signal Processing* 69 (2021), 500–514. https://doi.org/10.1109/TSP.2020.3046971
[62] Dimitris Stripelis, Paul M Thompson, and José Luis Ambite. 2022. Semi-synchronous federated learning for energy-efficient training and accelerated convergence in cross-silo settings. *ACM Transactions on Intelligent Systems and Technology (TIST)* 13, 5 (2022), 1–29.
[63] Toyotaro Suzumura, Yi Zhou, Natahalie Barcardo, Guangnan Ye, Keith Houck, Ryo Kawahara, Ali Anwar, Lucia Larise Stavarache, Daniel Klyashtorny, Heiko Ludwig, et al. 2019. Towards Federated Graph Learning for Collaborative Financial Crimes Detection. *arXiv preprint arXiv:1909.12946* (2019).
[64] Colin Tankard. 2016. What the GDPR means for businesses. *Network Security* 2016, 6 (2016), 5–8.
[65] Chunlin Tian, Li Li, Zhan Shi, Jun Wang, and ChengZhong Xu. 2022. HARMONY: Heterogeneity-Aware Hierarchical Management for Federated Learning System. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 631–645.
[66] Saeed Vahidian, Mahdi Morafah, and Bill Lin. 2021. Personalized Federated Learning by Structured and Unstructured Pruning under Data Heterogeneity. In *2021 IEEE 41st International Conference on Distributed Computing Systems Workshops (ICDCSW)*. 27–34. https://doi.org/10.1109/ICDCSW53096.2021.00012
[67] Ewen Wang, Ajay Kannan, Yuefeng Liang, Boyi Chen, and Mosharaf Chowdhury. 2023. FLINT: A Platform for Federated Learning Integration. (2023). arXiv:2302.12862 [cs.LG]
[68] Xinran Wang, Qi Le, Ahmad Faraz Khan, Jie Ding, and Ali Anwar. 2023. A Framework for Incentivized Collaborative Learning. *arXiv preprint arXiv:2305.17052* (2023).
[69] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv preprint arXiv:1804.03209* (2018). https://arxiv.org/abs/1804.03209
[70] Kang Wei, Jun Li, Chuan Ma, Ming Ding, Sha Wei, Fan Wu, Guihai Chen, and Thilina Ranbaduge. 2022. Vertical Federated Learning: Challenges, Methodologies and Experiments. arXiv:2202.04309 [cs.LG]
[71] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934* (2019).
[72] Yuexiang Xie, Zhen Wang, Dawei Gao, Daoyuan Chen, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. 2022. Federatedscope: A flexible federated learning platform for heterogeneity. *arXiv preprint arXiv:2204.05011* (2022).
[73] Hang Xu, Chen-Yu Ho, Ahmed M. Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis. 2021. GRACE: A Compressed Communication Framework for Distributed Machine Learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. 561–572.
[74] Jie Xu, Benjamin S Glicksberg, Chang Su, Peter Walker, Jiang Bian, and Fei Wang. 2021. Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research* 5, 1 (2021), 1–19.
[75] Wenyuan Xu, Weiwei Fang, Yi Ding, Meixia Zou, and Naixue Xiong. 2021. Accelerating Federated Learning for IoT in Big Data Analytics With Pruning, Quantization and Selective Updating. *IEEE Access* 9 (2021), 38457–38466. https://doi.org/10.1109/ACCESS.2021.3063291
[76] Chengxu Yang, Wang Qipeng, Mengwei Xu, Zhenpeng Chen, Bian Kaigui, Yunxin Liu, and Xuanzhe Liu. 2021. Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. In *The World Wide Web Conference*.
[77] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Trans. Intell. Syst. Technol.* 10, 2, Article 12 (jan 2019), 19 pages. https://doi.org/10.1145/3298981
[78] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903* (2018).
[79] Chaoqun You, Daquan Feng, Kun Guo, Howard H Yang, Chenyuan Feng, and Tony QS Quek. 2022. Semi-synchronous personalized federated learning over mobile edge networks. *IEEE Transactions on Wireless Communications* (2022).
[80] Sixing Yu, Phuong Nguyen, Waqwoya Abebe, Wei Qian, Ali Anwar, and Ali Jannesari. 2022. SPATL: Salient Parameter Aggregation and Transfer Learning for Heterogeneous Federated Learning. *To appear in the Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2022).
[81] Sixing Yu, Phuong Nguyen, Ali Anwar, and Ali Jannesari. 2023. Heterogeneous Federated Learning using Dynamic Model Pruning and Adaptive Gradient. arXiv:2106.06921 [cs.LG]
[82] Binhang Yuan, Song Ge, and Wenhui Xing. 2020. A federated learning framework for healthcare iot devices. *arXiv preprint arXiv:2005.05083* (2020).
[83] Jie Zhang, Song Guo, Zhihao Qu, Deze Zeng, Yufeng Zhan, Qifeng Liu, and Rajendra Akerkar. 2022. Adaptive Federated Learning on Non-IID Data With Resource Constraint. *IEEE Trans. Comput.* 71, 7 (2022), 1655–1667. https://doi.org/10.1109/TC.2021.3099723
[84] Sai Qian Zhang, Jieyu Lin, and Qi Zhang. 2022. A multi-agent reinforcement learning approach for efficient client selection in federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9091–9099.
[85] Tuo Zhang, Lei Gao, Sunwoo Lee, Mi Zhang, and Salman Avestimehr. 2023. TimelyFL: Heterogeneity-aware Asynchronous Federated Learning with Adaptive Partial Training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5063–5072.
[86] Xinwei Zhang, Wotao Yin, Mingyi Hong, and Tianyi Chen. 2021. Hybrid Federated Learning: Algorithms and Implementation. arXiv:2012.12420 [cs.LG]
[87] X. Zhang, X. Zhou, M. Lin, and J. Sun. 2018. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 6848–6856. https://doi.org/10.1109/CVPR.2018.00716
[88] Jiehan Zhou, Shouhua Zhang, Qinghua Lu, Wenbin Dai, Min Chen, Xin Liu, Susanna Pirttikangas, Yang Shi, Weishan Zhang, and Enrique Herrera-Viedma. 2021. A survey on federated learning and its applications for accelerating industrial internet of things. *arXiv preprint arXiv:2104.10501* (2021).
[89] Huanzhou Zhu, Bo Zhao, Gang Chen, Weifeng Chen, Yijie Chen, Liang Shi, Peter Pietzuch, and Lei Chen. 2022. MSRL: Distributed Reinforcement Learning with Dataflow Fragments. *arXiv preprint arXiv:2210.00882* (2022).

A Artifact Appendix

In this section, we give instructions for the evaluation of our artifact.

A.1 Abstract

The artifact is a code repository that contains the scripts and instructions for running our FLOAT (Federated Learning Optimizations with Automated Tuning) framework. FLOAT introduces an FL framework that uses adaptive accelerations to optimize resource utilization and model performance through a multi-objective Reinforcement Learning with Human Feedback (RLHF) mechanism. REFL uses FedScale framework (http://fedscale.ai) as the base for the implementation. FedScale provides a diverse set of datasets and benchmarks for FL training and evaluation with simulated resource and availability traces. To augment the realism of FedScale's simulations, we have incorporated real 4G and 5G client devices' network traces, enriching the framework's capability to mirror real-world conditions.

A.2 Description & Requirements

A.2.1 How to Access: The FLOAT code repository is available at https://github.com/AFKD98/FLOAT. It includes scripts and instructions for setup, installation, and running experiments.

A.2.2 Hardware Dependencies: Running experiments do not mandate any special hardware. However, to run the experiments in a reasonable amount of time servers with fast Nvidia GPUs (e.g., A100/V100) or at least 3070 GPUs are recommended. However, due to the scale of the experiments conducted in this study, it may not be feasible to reproduce it due to the large cost incurred. To give an estimate, even with advanced GPUs such as RTX 3070 GPUs, it took a significant amount of time to run them (i.e., 1400 hours of GPU time). This makes it quite hard to reproduce the claims/figures within the time frame set for evaluation.

A.2.3 Software Dependencies: The FLOAT framework's operation requires Python for core programming, Anaconda for package and environment management, and CUDA for GPU support in accelerated computing tasks. Essential packages and libraries required for FLOAT are included in the **environment.yml** file and **requirements.txt** within the FLOAT repository.

A.2.4 Benchmarks: FLOAT supports various FL tasks including image classification and speech recognition. Other tasks such as language modeling can also be added as they are supported by FedScale and FLOAT leverages FedScale's extensive dataset and benchmark suite. To provide a realistic simulation environment, FLOAT includes real-world traces for compute, network, and client availability. These traces are critical for accurately simulating FL environments and are located in the following directory of FLOAT's GitHub repository:

`FLOAT/benchmark/dataset/data/device_info/`

Details of the traces are provided in Section 6.1.

A.3 Setup

A.3.1 Installation: NOTE: Although FLOAT automatically finds the correct paths using `os` commands in Python. Nevertheless, please ensure that the paths to the code and datasets are consistent across all nodes so that the simulator can find the right path. FLOAT can run on a single node or on multiple nodes that have at least a 3070 Nvidia GPU and the CPU capacity to run at least 30 threads in parallel.

Quick start: After cloning the repo, go to the main directory FLOAT using the following command:

```
cd FLOAT
```

First, edit **float_install.sh** script if necessary. Please, uncomment the parts relating to the installation of the Anaconda Package Manager, CUDA 10.2 if they are not already present on the servers. Note, if you prefer different versions of conda and CUDA, please check the comments in **float_install.sh** for details. After editing, run the following commands to prepare the environment:

```
source float_install.sh
```

Running **float_install.sh** should install all dependencies in a conda environment and also activate the `fedscale` conda environment on the bash terminal.

Manual install:

Alternatively, the installation and setup can also be done by running the following commands also provided in the README in order:

```
cd FLOAT

# Please replace ~/.bashrc with ~/.bash_profile
#for MacOS
FLOAT_HOME=$(pwd)
echo export FLOAT_HOME=$(pwd) >> ~/.bashrc
echo alias fedscale=\'bash $FLOAT_HOME/float.sh\'
>> ~/.bashrc
echo alias float=\'bash $FLOAT_HOME/float.sh\'
>> ~/.bashrc
conda init bash
. ~/.bashrc

conda env create -f environment.yml
conda activate fedscale
pip install -r requirements.txt && pip install -e .
```

Ensure that the `FLOAT_HOME` environment variable is set, and the `fedscale` and `float` alias are configured in `/.bashrc` as per the instructions in the repository.

A.3.2 Setting the Environment Variables: It is crucial to set specific environment variables for the experiments in the `/.bashrc` and config files. These variables ensure the correct execution paths and configurations are utilized during the runtime. The required environment variables are:

- `FLOAT_HOME` - The root directory of the FLOAT installation.
- `CONDA_PATH` - The path to the Conda source script, typically `$HOME/anaconda3/etc/profile.d/conda.sh`.
- `CONDA_ENV` - The name of the Conda environment used for FLOAT, which should be activated before running experiments.
- `fedscale` and `float` aliases for execution script locations.

A.3.3 Experiment Configurations: All config files are located in FLOAT/benchmark/configs directory. The repository includes config files for all our experiments in the paper. The config files include cluster configuration, job parameters, and model specifics necessary for initiating the training process. New configs can be added for adding more datasets and models from FedScale.

To streamline the setup and execution of the experiments ensure you tailor the configurations to fit your system. This involves adjusting dataset locations, model parameters, learning rate, batch size, and the number of training rounds. These adjustments are crucial for optimizing the experiment's efficiency and reproducibility, making it more accessible for both researchers and practitioners.

A.3.4 Running Experiments: NOTE: Running experiments requires `SSH` access for launching and communication between the aggregator server and clients. Please make sure to have conda environment `fedscale` activated. FLOAT's experiment manager in **FLOAT/float_run_exps.sh** automatically checks if the dataset is downloaded, otherwise, it downloads the dataset required to run. Alternatively, each dataset can manually be downloaded as well using the following command inside the main FLOAT repo directory:

```
cd FLOAT
./benchmark/dataset/download.sh download DATASET
```

[Preparation]

Replace `DATASET` with the dataset required for the experiment. An example command for the FEMNIST dataset is:

```
./benchmark/dataset/download.sh download femnist
```

[Execution]

To execute an experiment, use:

```
bash float_run_exps.sh -d dataset -a algorithm
```

For example for running Oort this is the command:

```
bash float_run_exps.sh -d femnist -a oort
```

and for running Oort with FLOAT the command is as follows:

```
bash float_run_exps.sh -d femnist -a oort_float
```

More commands for running other experiments are also provided in the README of the repository. Alternatively, they can be accessed by running:

```
bash float_run_exps.sh -h
```

[Resluts]

All experiments should produce logs in the directory FLOAT/benchmark/logs/ and aggregated logs in the main FLOAT directory as `femnist_logging` if the dataset used is FEMNIST.

These experiments should validate the claims above by comparing FLOAT's performance against baseline FL frameworks on tasks like image classification or speech recognition, using the provided benchmarks in the paper.

A.4 Evaluation Workflow

A.4.1 A.4.1 Example Claims: Following are the major claims made in our paper: In general, FLOAT with any client selection algorithm should reduce the amount of dropped clients and resource wastage while preserving accuracy and also showing accuracy improvement under highly heterogeneous data conditions with low system resources. The logs should provide this information at the granularity of per round.

A.4.2 Additional experiments: Additional experiments related to resource variations can be automatically verified while running the experiments as the logs will contain resource information of clients. The variations can also be controlled by modifying the functions:
`get_completion_time_with_variable_network` and
`get_new_network_bandwidth` in
FLOAT/fedscale/cloud/internal/client_metadata.py. In addition, results in Figure 9 and 10 can be generated by the analysis of Q table of the RLHF agent that can be fetched by executing:

```
python FLOAT/benchmark/logs/rl_model/load_Q.py
```

A.5 Notes on Reusability

FLOAT supports extension with new FL tasks, datasets, and acceleration techniques. It is designed to be non-intrusive and compatible with existing FL systems, facilitating easy integration and experimentation.