Beyond FLuID: Exploring Sparsity and Algorithm Variability in Federated Learning

Arati Ganesh Aditya Iyer Panchami Kamath Faris Qadan

ABSTRACT

Federated learning presents a novel approach to training machine learning models on distributed mobile or edge devices, preserving user privacy by locally performing training with shared model parameter updates to a central server. However, heterogeneous device capabilities can lead to straggler devices, impeding overall training time and model accuracy. In this work, we replicate and extend prior research on leveraging Invariant Dropout in federated learning. To begin, we recreate FLuID's results in a simulated setting, making it suitable for use without physical devices in cost-critical scenarios. We then extend FLuID in three sequential stages. Firstly, we benchmark Invariant Dropout's performance in conjunction with FedAdam compared to FedAvg. Secondly, we implement both raw and scaled sparsity thresholds to further prune the submodels being sent to straggling clients. Thirdly, we quantize previously unquantized layers in the aforementioned submodels. In all cases, the effects of these changes on the FLuID framework's performance is analyzed from the lenses of accuracy and convergence rate. The experiments performed showed FedAdam to be a strong competitor to FedAvg alongside Invariant Dropout, despite its higher energy costs. Furthermore, tight sparsity thresholds (in both the raw and scaled cases) proved to be useful optimizations to FLuID. Finally, quantization in general was shown to hurt model accuracy.

1 INTRODUCTION

Federated learning (FL) is a novel approach whereby machine learning models can be trained on mobile and edge devices, with each device storing data locally. This is fundamentally different from how machine learning is typically done where all the data is stored on a central server before the training process. This addresses user privacy and security concerns as raw data is not sent through the network and enables training to be performed on a diverse set of mobile and edge devices that are capable of generating and storing their own data.

Despite this, the main disadvantage of FL is the fact that training time is always a function of the slowest device in the cluster or training setup, also known as the "straggler" device. In any real-world setup, it is expected that different devices have different performance characteristics and the training process can have varying effects on different devices. Several studies have been done to tackle this problem [10, 12]. These studies introduce asynchronous training paradigms where individual clients can asynchronously train the global model on the central server, meaning that devices with fast training times will update the global model faster while stragglers will still be training older data. Others [1, 8] prove that this approach could slow down model convergence and potentially reduce model accuracy.

Mahmoud et al. [25] introduce an entirely new approach where stragglers are eliminated from the training process altogether based on the computation and communication latency of the device in comparison to a total latency constraint. This has the potential for introducing arbitrary latency constraints and also introduce training bias where valuable training data from straggler devices are discarded solely due to the performance characteristics of the devices themselves.

Recent research aims to use a technique called dropout, that would reduce the computational intensity on clients. Wen et al. [36] introduce a technique called Federated Dropout, where each iteration reduces the model to a "subnet", which the server then redistributes across clients for updating. The size of the subnet as well as the dropout rate of the device is adapted at runtime based on the communication and computation constraints of individual devices. However, this technique uses Random Dropout, meaning that the model accuracy takes a slight hit due to (~1% on a relatively small dataset).

The FLuID [35] framework on the other hand, demonstrates how all the deficiencies previously described are addressed or mitigated. FLuID uses Invariant Dropout, where neurons immune or "invariant" to further training iterations are dropped. This ensures that only neurons that contribute positively to the training of the global model are used. To account for runtime performance degradation of clients due to network or local issues, FLuID ensures that the submodel generated after Invariant Dropout is periodically calibrated. FLuID can offer a performance speedup of up to 18% on real-world devices and also improve model accuracy over other state-of-the-art dropout techniques.

As federated learning continues to evolve, the integration of quantization and sparsity into frameworks like FLuID highlights a strategic move towards optimizing computational resources and network efficiency. Currently, FLuID does not support optimizations of this nature. Adding support to take advantage of these characteristics plays into the

goals of Wang et al. by allowing for more extensive mitigation of the communication bottlenecks that typically hinder FL [35].

The FLuID framework incorporates these techniques to mitigate the adverse impacts of device disparity. By selectively pruning and quantizing model components, FLuID ensures that only the most critical information is processed and transmitted, thereby reducing the latency and overhead associated with training large models across decentralized networks.

1.1 Key Contributions

The primary contribution of this project is to recreate the FLuID framework in its entirety using custom devices built with the in-house cluster infrastructure, that mimic real-world mobile and edge devices.

In brief, this enables the use of the framework in the absence of physical mobile devices, using simulated CPU and/or GPU devices to carry out federated learning. While the FLuID framework was tested on 5 clients, our infrastructure allows us to scale the number of clients to 20 times that value thereby allowing us to perform a more comprehensive performance evaluation using larger models.

Additionally, we formally benchmark FLuID's performance with FedAdam [32] against FedAvg with regards to training accuracy.

Finally, we propose two unique optimizations specific to Invariant Dropout: one whereby submodels are further pruned by raw or scaled sparsity thresholds before being sent to client devices, and one whereby fully-connected (FC) layers are quantized to further alleviate the communication and computation bottlenecks that hinder federated learning.

We test these optimizations and recommend which may be suitable additions to Invariant Dropout in future iterations of the work.

2 BACKGROUND AND MOTIVATION

Federated learning (FL) is a way to collaboratively train a model without having to share data that will be used to train parts of the model. This provides motivation to use federated learning for a wide variety of applications where privacy is of priority and data sharing is not possible or permitted. This also prevents the need to store and share data on cloud for training the model.

One barrier-to-entry that FL generally presents is the fixed and variable costs involved in physical device acquisition and maintenance. This is particularly relevant in testing and prototyping phases. These costs come in the form of purchasing these devices, as well as ongoing costs involved in charging and maintaining them over time.

In the case of developing FL models in cost-critical situations, these are large expenses that are difficult to amortize. This emphasizes the need to make FL, and specifically the FLuID framework, easy-to-simulate.

More generally, only the updates with respect to the model is shared by the devices involved. Thus federated learning helps to collaborate between a variety of sources for data, which in turn helps in training and development of more robust and generalized models for a variety of applications.

The first time FL was proposed was in the year 2016 by Google [3]. It was mainly then used for Android Devices to contribute to models locally without causing privacy concerns. More recently, there has been research in various fields using federated learning like smart medical systems, smart city applications, finance, and more.

Even with such wide variety of applications and advantages, there are certain challenges that using FL poses. These are primarily observed by way of privacy preservation concerns, communication overhead and heterogeneity in the client devices that contribute to model training. The wide range of applications provides the motivation to resolve these challenges that arise when using FL.

Out of all the challenges mentioned, heterogeneity can considerably affect the scalability and communication capabilities in federated learning. Difference(s) in the training and computing capacity of some of the clients can affect the training times of the model as traditionally the server has to wait to get the model updates from all of the clients involved. Such devices that significantly reduce the run time of the algorithms are known as stragglers.

There are many works that propose straggler resilient algorithms to address this issue [33]. Model dropout techniques are used to balance load between stragglers and non stragglers. Existing works like Federated Dropout [7], Ordered Dropout [17], propose methods for dropping neurons in the model for creating submodels. Federated Dropout performs dropout randomly and clients train random subsets of the model.

While this showed promising results for scaling up the model, it had a negative effect on the accuracy. Ordered Dropout performs dropout systematically and gives better accuracy as compared to Federated Dropout however when compared to vanilla federated learning there is a drop in accuracy [35]. Thus, there is significant scope to improve dropout techniques to improve overall accuracy. For this project in particular we plan to implement Invariant Dropout methodology for federated learning proposed in the FLuID paper. This dropout technique performs better than Ordered Dropout across the different datasets that were used to test FLuID [35].

Apart from dropout techniques, there are various optimizations that can be done in the federated algorithms themselves.

Federated Average algorithm is a popular FL algorithm that performs multiple updates locally on the client before it communicates with the server thus reducing the communication bottleneck. However, there is evidence that its performance may not be up to mark in the case of heterogeneous clients and causes slow and unstable convergence [20].

Thus, current research efforts aim to improve the efficiency as the heterogeneity poses a significant challenge in FL. Upon successful implementation of Invariant Dropout technique, we thus propose implementing FLuID with FedAdam [32]. This is an adaptive optimization algorithm that avoids many of the convergence-related pitfalls of FedAvg [20].

Network instability and capacity are key considerations in wireless communications and this provides scope for research of communication efficient FL algorithms. A number of works deal with model pruning and sparsity to deal with communication overhead and unstable networks [26]. Thus, we also wish to explore the effect of sparsity in federated learning to improve communication efficiency and reduce any negative impact due to stragglers.

Sparsity is a key characteristic prevalent in many models today. It allows for significant performance and efficiency improvement in both training and inference of machine learning models [16]. In short, it involves the 'inorganic' decrease in size of model weights by zeroing out very small weights. The threshold that defines 'small' is defined on a case-by-case basis, and can either be unstructured and guided by specific heuristics, or more structured and formally derived [22].

Quantization is a method of further reducing a model's training load on its clients. It does so by (typically) reducing the precision of model weights, effectively leading to a somewhat pruned model.

3 RELATED WORK

With regards to simulating the federated learning process, Flower [2] enables simple simulation of CPU or GPU devices. This is done by simply partitioning available compute resources and splitting them according to need and demand.

FL_Pytorch [4] performs a similar function alongside a user-friendly interface. However, FLuID [35] is built atop the aforementioned Flower library, making it a more natural contender for a simulation-based recreation of the framework.

Previous studies have addressed straggler issues in heterogeneous client setups. Dropout methods, like Federated Dropout [7, 31], randomly drop segments of the global model to distribute submodels to slower devices, potentially affecting accuracy.

Recent advancements, such as Ordered Dropout [11, 18], aim to mitigate accuracy loss but overlook individual neuron contributions. Server offloading techniques, including those by [37] and [15], involve offloading part of the model to servers or transferring knowledge to larger server-side CNNs.

Communication optimization strategies, like [5, 9, 10], aim to reduce learning time. Model pruning methods, such as PruneFL [19] and approaches by Mounir et al. [29], generate either a single or static submodel, limiting contributions from slower clients. Coded federated learning techniques, like those introduced by Schlegel et al. [34], introduce redundancy to mitigate the impact of slower devices. Unlike these, Invariant Dropout allows clients to retain their data locally.

SuperFed [21] co-trains multiple models, sending subnetworks of different sizes to all clients, while FLuID focuses on optimizing stragglers' performance by training most clients on the full global model and a smaller percentage on tailored submodels.

Other works have explored various variants of federated learning beyond the baseline FedAvg algorithm. One such work, by Reddi et al. [32], proposes FedAdam. This is an adaptive FL optimization algorithm that offers more favorable convergence bounds than FedAvg. The outlook of this is a boost in communication efficiency and robustness, outperforming FedAvg in federated learning performance despite being slightly more compute-intensive.

With regards to sparsity, FLARE [14] represents a breakthrough in federated learning by introducing a sparsity-focused approach that reduces communication needs. By selectively transmitting essential updates and accumulating others locally, coupled with error correction and regularization techniques, it effectively minimizes delays in updates.

The gap lies in existing methods' oversight of individual neuron contributions and their inability to dynamically adjust dropout rates accordingly. Invariant Dropout aims to address this gap by introducing a novel approach that prioritizes neuron importance, leading to optimized model training, particularly in heterogeneous client setups. Building on this, our proposal includes integrating the FLuID framework with FedAdam and benchmarking it against its current state, using FedAvg. We also look to explore the effects of sparsity and quantization on FLuID's efforts to widen communication and computation bottlenecks in federated learning.

While sparsity in isolation (or rather, with conventional deep learning applications) is very well-documented ([16], [27]), this is not explicitly true in conjunction in federated learning settings. In the non-FL case, it has been shown to have a positive impact on model generalizability and accuracy [16], while providing relatively easier training and inference workloads.

Mao et al. proposed SAFARI [26] as a successful method of alleviating communication bottlenecks in FL. This, however, is done in a dropout-less setting, whereby the only submodel pruning to occur comes by way of aggressive sparsity threshold implementation.

The same can be said about quantization. Its effects on general deep learning models are very well-documented [13].

Liu et al. propose 'Hierarchical FL' [23]. This is a model utilizing quantization to prune neural networks and demonstrate its effectiveness in mitigating the effects of communication bottlenecks during the federated learning process.

4 SYSTEMS AND ARCHITECTURE DESIGN

FLuID [35] in its public state (both code and paper) is presented for use with physical Android devices. As such, we propose accomodations for simulating client devices in cost-critical cases where these physical devices are not readily available.

This is made simple with the Flower [2] library in Python. We use this library to simulate n client CPUs at a time in training.

Specifically, Flower's virtual client engine (NumPyClient) enables for the partitioning of a single processor's resources (ideally a GPU) for simulation of as many CPU devices as required.

Given the scale of our experiments to follow, one single GPU suffices in this scenario. Our experiments do not simulate more than 8 clients at a time, making this problem relatively tractable and less resource-intensive.

The advantages of our system manifest in its ability to simulate a considerable number of clients concurrently, demonstrating scalability and cost-effectiveness. As our objective centers on extending FLuID [35] by investigating diverse optimization methodologies and their impact on accuracy, there is no imperative to conduct tests using physical devices. Although it may be contended that virtual clients offer high consistency and pose challenges in straggler identification, each client still receives dynamically allocated resources.

The primary limitation lies in the consistent performance of individual clients across epochs, potentially leading to the recurrence of the same virtual client as the straggler in each iteration. However, this does not impede our experiments, which aim solely to evaluate various optimizations atop FLuID [35] in a cost-effective manner.

5 EVALUATION

Given the broad, multi-faceted nature of this project, consistent reflection and evaluation of progress will be vital in ensuring its success.

5.1 Evaluating Project Success

The project's overall success can be loosely defined by the success of its three distinct stages. As such, we set an (a) goal, a (b) goal, and a two-pronged (c) goal in order of tractability and importance to the overarching project motivation:

(a) Complete recreation of FLuID [35] results using CPU/GPU-simulated counterpart
(b) Formal accuracy bencmarking of FLuID [35] simulation with FedAdam [32] and FedAvg [20]
(c) Successful addition and testing of optimization to use sparsity and quantization in mitigating effect(s) of stragglers in FL

While these three tasks all present significant challenges independently, accomplishing any (or all) of them would represent great strides towards fulfilling the motivations of this project.

Since any additional simulations and optimizations will be built atop a simulated version of FLuID, goals (b) and (c) are bottlenecked by the completion of goal (a). This further affirms the importance of looking to complete the above goals in sequence.

5.2 Evaluation Metrics and Baselines

Each of the above goals requires a unique set of baselines to be effectively evaluated and deemed complete. The success of all three goals is primarily underpinned by the success of the first.

To evaluate the completion of goal (a), we compare both the accuracy and computational performance of our developed simulation against those produced in the FLuID paper [35]. Specifically, we compare our simulation's performance to FLuID's when trained on the FEMNIST [6] dataset.

Training time and trained model accuracy will be found simply by aggregating time taken to train each model, and a weighted average of distributed accuracy and loss per client, as proposed by the authors of FLuID [35].

It must be noted that in a federated learning environment, each training device incurs its own accuracy measurement. As such, in aggregating final measurements, a simple arithmetic mean will be applied to compare implementations.

Finally, our own models' performances will be aggregated against FLuID's by calculating the relative percentage difference between the time and accuracy measurements obtained. We will consider a difference of ±10% in accuracy measurements to be a success, and a percentage increase or decrease when differing model factors in sequence of ±5%. That is, given that the nature of our training clients will differ from those used in the paper, our simulation aims to reflect the general trend of the impact of modifying different FL factors on training time, as opposed to the training time itself.

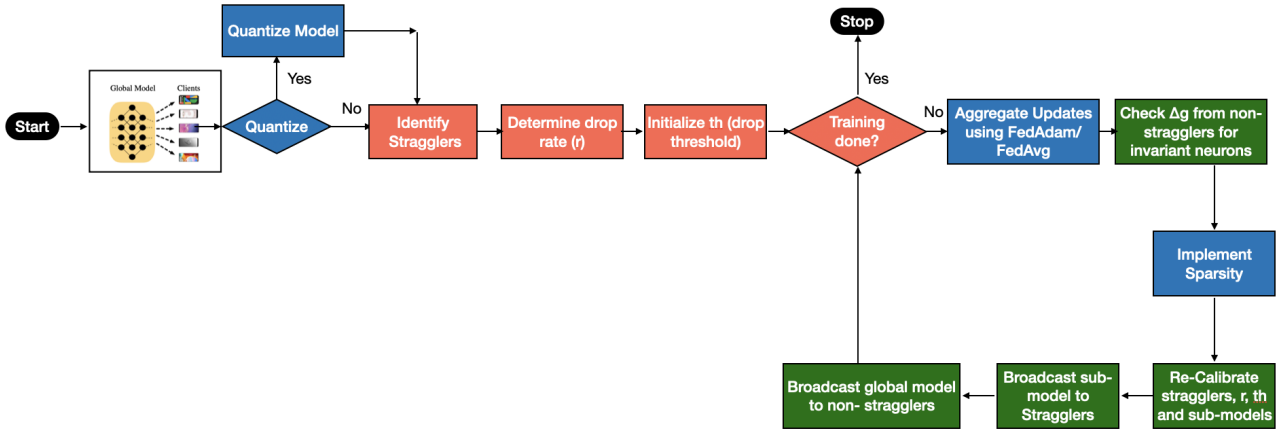

Figure 1: Overview

To evaluate goal (b), we undergo a similar set of operations. We measure training time and accuracy as detailed previously for each implementation of our simulator using a different FL algorithm. In the case of goal (b), however, we ultimately rank the performance of each simulation through a weighted sum of the training time and accuracy measurements, proposing a 'best' solution given various unique use-cases.

Finally, similar logic holds in evaluating goal (c). Here, we compare the training time and model accuracy measurements of our now-optimized simulation to that of our vanilla FLuID simulation, achieved by completing goal (a). The overall performances of the two are then compared similarly to above.

5.3 Systems for Subgoal Evaluation

The simulations and subsequent evaluations will be developed and performed using Python, with the FL model itself being built using PyTorch modules [30].

As mentioned previously, the FEMNIST dataset [6] will be used for benchmarking and evaluation of model accuracies given different changes and optimizations implemented.

One key difference in our proposed implementations and those documented by Wang et al. [35] comes in the difference in client device access. In the paper, these come in the form of a variety of Android devices.

In the case of this project, Georgia Institute of Technology-provided PACE-ICE compute clusters will be used to simulate clients of similar CPU specifications to those listed in Table 1 of the FLuID paper [35].

Furthermore, in the case that training times become excessively high given the allocated timeframe, GPUs, also generously provided on GT's PACE-ICE servers, may be utilized.

5.4 Testing Methodologies

As mentioned previously, the success of this work can be measured in a multi-phase format. Specifically, by the sequential accomplishment of the (a), (b), and (c) goals stated.

More tangibly, testing and recreating all the paper's results is somewhat intractable, especially given the month-long time-span involving these numerous goals. As such, a subset of the overall test space is to be carried out.

The success of the (a) goal will be measured by testing the proposed recreation against the FEMNIST [6] dataset. Given its relative simplicity, it enables testing for the aforementioned metrics a trivial task.

To further promote this feasibility, GPUs will be used in the device simulations, as opposed to CPUs. This allows for longer-term testing in a far shorter actual time.

Following this, FL optimization algorithms like FedAvg and FedAdam will be compared, as mentioned before. The testing of these algorithms will also be performed using the FEMNIST [6] dataset. Its simplicity, again, makes the task of evaluating three separate models comprehensively a tractable task. As was previously mentioned, tests will

compare the training accuracies, training times, and computing efficiencies of these models with each of the different algorithms used.

Finally, to evaluate the impact of sparsity-based optimizations on FLuID [35], the FEMNIST [6] dataset will again be used. This time, however, the framework's performance will be compared against the 'vanilla' FLuID recreation (GPU device simulations, FedAvg optimization) in the same relevant metrics. This will enable a comparison of the paper's results with the proposed extension(s) directly.

5.5 Experiments and Results

The following subsections discuss various experiments with respect to Federated Learning Optimization Algorithms, Sparsity and Quantization along with their respective results.

5.5.1 ***Optimization of Federated Learning Algorithms.*** The recreation of FLuID with the FedAvg was done using the Flower [2] library in Python. As detailed previously, this library enables accessible simulation of CPU or GPU devices for federated learning.

The first phase of testing involved simply recreating the results of FLuID [35] in the simulated setting. To begin, we evaluate the performance of the vanilla recreation of FLuID in a simulated setting. This was run over 200 rounds, using 5 simulated clients. That is, one client is classified as a straggler and dropped each round.

Training time (seconds)	Average accuracy (%)
153	68.32

Table 1: Training time and average accuracy for FLuID recreation over 200 rounds with 5 clients.

As is reflected in Table 1, this is in line with the results presented by the authors of FLuID, within a 15% margin of error. This error can be attributed to differences in device configuration and hyperparameters that were kept unmodified in this case.

Given these results, the recreation is evaluated comparing FL optimization algorithms. To evaluate the effect of using different optimization strategies on the effectiveness of the FLuID framework, a staggered approach was taken. That is, tests were performed to test the effectiveness of FLuID with these strategies in three different scenarios:

- Limited time, limited device availability (100 rounds and 5 clients)
- Limited time, extended device availability (100 rounds and 8 clients)
- Extended time, extended device availability (200 rounds and 8 clients)

This was done in order to benchmark the capabilities of the framework in various, close-to-real-world situations, given different constraints that may face researchers.

The results comparing the performance of FLuID with FedAvg and FedAdam algorithms can be found in the following tables.

Model spec. \ Optim. strat.	FedAvg	FedAdam
5 clients, 100 rounds	72	75
8 clients, 100 rounds	77	88
8 clients, 200 rounds	153	162

Table 2: Training time (in seconds) while varying optimization strategies and model specifications.

Model spec. \ Optim. strat.	FedAvg	FedAdam
5 clients, 100 rounds	68.32	69.95
8 clients, 100 rounds	57.36*	71.84
8 clients, 200 rounds	72.42	75.91

Table 3: Average model accuracy (%) while varying optimization strategies and model specifications.

From the above tables, we arrive at various conclusions regarding the relation between using other optimization strategies as opposed to the simple FedAvg in conjunction with FLuID. Firstly, in the general case, FedAdam (and likely most other more complex optimization algorithms) are more time-consuming than FedAvg.

In the most drastic case observed in the results in Table 2, using FedAvg presents a 14.3% speedup on FedAdam with FLuID. This comparison is made when simulating federated learning on GPUs. As such, the difference is likely to be magnified (and even amplified) when training on conventional mobile devices.

This finding follows the work of Reddi et al. [32], given that FedAdam requires the processing of slightly more parameters. While this makes the algorithm more versatile and averse to sudden change, it is likely to be the cause of the slightly increased training time above.

With regards to model accuracy, we see that with the exception of one likely anomalous* datapoint, both FedAvg and FedAdam present almost identical results with negligible discrepancies.

It should also be noted that for a larger number of clients, the convergence rate increased. Briefly, this indicates the increased effectiveness of invariant dropout in mitigating the effect of stragglers as the number of stragglers (and of course, overall clients) increases.

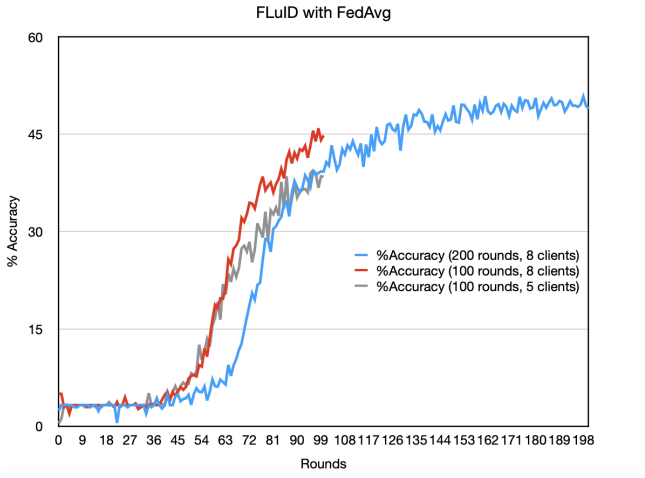

Figure 2: Accuracy convergence of FLuID w/ FedAvg

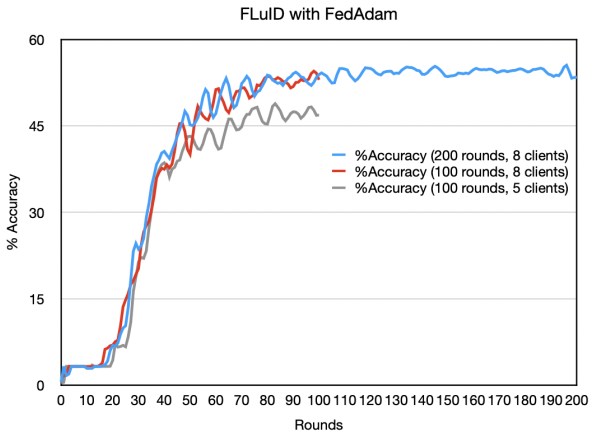

Figure 3: Accuracy convergence of FLuID w/ FedAdam

This is likely due to FedAdam's position as a 'descendant' of FedAvg, whereby the former algorithm simply extends the latter. Given the trivial nature of training task at hand, the differences between the two algorithms are negligible.

Figures 2 and 3 allow for the visual affirmation of the aforementioned deductions. As shown, the final accuracy measurements converge to similar points, with FedAdam displaying a *slightly* ($\sim$ 10%) higher accuracy reading.

As such, these deductions reaffirm FedAvg as a sound companion algorithm for invariant dropout.

While FedAdam presents a slight improvement in training accuracy, the returns diminish over many rounds, particularly in comparison to the increased training time.

The primary, underpinning conclusion from these preliminary results is that FLuID performs at a similar level

Algorithm 1 FedAdam

Require: β_1: Momentum parameter, β_2: Second moment parameter, η: Server-side learning rate, τ: Degree of adaptability, η_l: Client-side learning rate
Ensure: Optimized weights **W_opt**

$FedAvg\_weights \leftarrow FLuID\_with\_FedAvg\_Weights$
for x,y in FedAvg_weights,current_weights **do**
  $\delta_t = x - y$
end for
for x,y in m_t, δ_t **do**
  $m_t = (\beta_1 * x) + (1 - \beta_1) * y$
end for
for x,y in v_t, δ_t **do**
  $v_t = (\beta_2 * x) + (1 - \beta_2) * y^2$
end for
for x,y,z in $current\_weights, m_t, v_t$ **do**
  $new\_weights = x + \eta * \frac{y}{\sqrt{z}+\tau}$
end for
$current\_weights \leftarrow new\_weights$
W_opt $\leftarrow$ current_weights

regardless of the optimization algorithm. The framework, even unoptimized for to take advantage of sparsity as is to implemented in the final phase, is applicable to many situations, both constrained and not-so constrained.

5.5.2 ***Quantization***. In our study, we employed dynamic quantization via PyTorch's quantize_dynamic module to convert a floating-point model into a dynamically quantized model. This approach is particularly adept at handling models that include a mixture of layer types, such as Convolutional 2D layers, Rectified Linear Units (ReLUs), Linear layers, and Long Short-Term Memory (LSTM) layers. The quantization process selectively applies weight-only quantization, which is particularly beneficial for layers that possess large weights, such as Linear and RNN layers. For our model, this method resulted in the quantization of the Linear layers only.

The quantization process was executed on two data types: float16 and int8. The experimental analysis involved 100 rounds of computation distributed across 5 client systems. This setup was designed to test the robustness and efficiency of the quantization process under realistic distributed computing scenarios. The outcomes of these experiments were recorded and plotted to assess the impact of quantization on model performance and size as shown in Figure[4]. Notably, the quantization to int8 demonstrated a discernible difference in accuracy compared to the original floating-point model and the float16 quantized version.

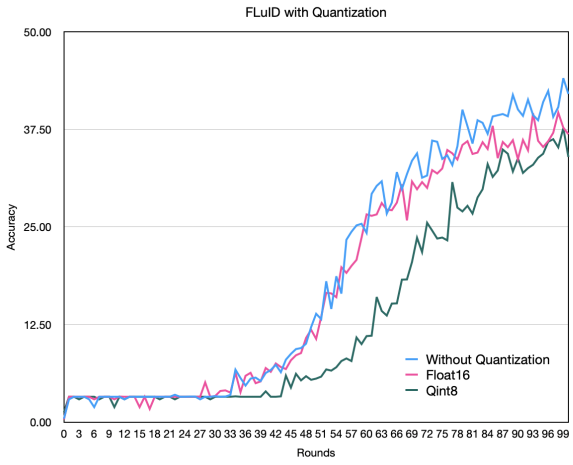

Figure 4: Accuracy vs Number of rounds for different precision types used for quantization

In terms of model size reduction, the original unquantized model had a size of 1,642,920 bytes. Post-quantization, the model size was significantly reduced to 493,394 bytes when using int8 quantization. This represents a substantial decrease in model size, which underscores the effectiveness of int8 quantization in optimizing models for environments with stringent memory constraints.

The reduction in model size achieved through int8 quantization comes at the cost of a slight degradation in accuracy. This trade-off highlights a critical consideration in the deployment of quantized models, especially in resource-limited settings where model performance must be balanced against memory and computational efficiency. The results suggest that dynamic quantization, particularly using int8, offers a viable path for deploying complex neural network models on edge devices, where memory and power are limited.

5.5.3 ***Sparsity***. The FLuID framework, as previously outlined, employs Invariant dropout, a technique that discards weights unchanged after a specified number of epochs, thereby reducing the computational load on straggler devices. These stagnant weights typically offer minimal contribution to training the global model. To discern such invariant neurons, FLuID leverages non-straggler devices to dynamically establish a threshold for identifying them. A pivotal aspect of this strategy lies in the adaptive selection of the threshold value. Although a test setup with five devices effectively demonstrates functionality, it fails to address scalability concerns. In real-world scenarios, where Federated Learning may encompass hundreds of clients concurrently, the computational cost associated with dynamically computing the invariance threshold at each epoch warrants scrutiny. Should this overhead prove substantial, it could undermine the primary objective of mitigating stragglers.

To investigate this, we explored alternative approaches to alleviate the burden on straggler devices. As noted in literature, L1 (Lasso) regularization has long been instrumental in inducing sparsity in machine learning models [24, 28]. The formulation of L1 regularization (Equation 1) dictates that coefficients or parameters in a model are penalized more significantly based on the regularization strength (λ), thereby driving them closer to zero. This concept of sparsity entails a model wherein a fraction of parameters are zero or close to zero, thereby reducing computational overhead during training.

To discern the distinction between sparsity and invariant dropout, or dropout in general, an analogy can be elucidative. Consider a factory housing a number of machines engaged in manufacturing. Operational constraints necessitate intermittent cessation of operations for certain machines. This strategy, akin to dropout, entails probabilistically shutting down machines at specified intervals, resulting in a reduced workload. Conversely, sparsity entails identifying and selectively reducing the workload of underperforming machines, analogous to pruning parameters in a model. Sparsity has been leveraged in deep learning to reduce model size, as corroborated in prior studies. Mao et al. [26] extended this concept to Federated Learning with unreliable communication, implementing an algorithm wherein clients employ masks to prune and update models before transmitting them to the server. The server, in turn, assesses model similarity across clients and replaces updates from unresponsive clients with those from the most similar peers. While akin to straggler mitigation, this approach differs in its replacement of models from unresponsive devices with those from similar peers.

In our framework, we maintain the working assumption of all devices participating in training, albeit at varying speeds. Implementing sparsity involves identifying parameters from straggler devices and applying a thresholding function. We explore two approaches: raw and scaled thresholding, described in Algorithm 2.

$$\text{Loss} = \text{Original Loss} + \lambda \sum_{i=1}^{n} |w_i| \tag{1}$$

To rigorously evaluate our algorithm, we established a testing environment comprising five GPU clients. Leveraging the straggler identification algorithm delineated in FLuID, we subsequently applied our thresholding algorithm to the weights of identified straggler devices. The simulation encompassed 200 rounds, during which we systematically varied the threshold values (τ) for both the raw and scaled threshold algorithms. Subsequently, we plotted the accuracy

Algorithm 2 Thresholding for Model Sparsification

Require: **W**: weights, n: number of weights, τ: threshold multiplier, *thresholding_type*: 'raw' or 'scaled'
Ensure: Sparse weights **W'**

```
max_weight ← max{W_0, W_1, ..., W_n}
if thresholding_type is 'scaled' then
    τ ← τ × max_weight
end if
for i ← 0 to n do
    if W_i < τ then
        W_i ← 0
    end if
end for
W' ← sparse weights after thresholding
```

against the number of rounds to discern the optimal τ value. Figures 5 and 6 depict the accuracy trends for raw and scaled thresholding, respectively.

Our analysis reveals that, across both thresholding methods, a τ value proximal to 0.01 yields the highest accuracy. However, alternative threshold values fail to yield high accuracy in the case of raw thresholding. Conversely, with scaled thresholding, we observe a gradual decline in accuracy with higher scaling factors. As anticipated, we observe a steep decline in accuracy with increasing threshold values, indicative of the inadvertent omission of crucial weights from the training process. Although our approach achieves commendable accuracy, it falls short of the performance attained by FLuID. This disparity can be primarily attributed to the fixed nature of our threshold values throughout the simulation, particularly evident in the case of raw thresholding. While scaled thresholding introduces a degree of adaptability by scaling the threshold based on the maximum weight value, the scaling factor itself remains constant. Nevertheless, despite these limitations, our approach exhibits promising accuracy levels while significantly mitigating computational overhead compared to FLuID.

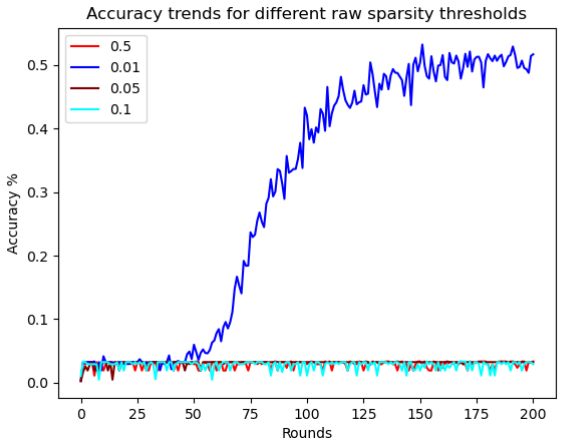

Figure 5: Accuracy vs Number of rounds for different values of raw sparsity threshold

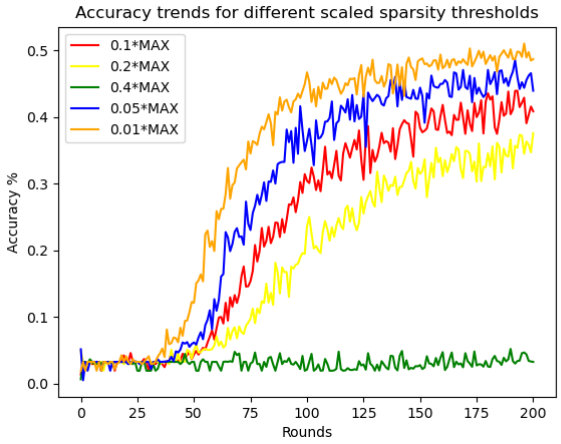

Figure 6: Accuracy vs Number of rounds for different values of scaled sparsity threshold

6 CONCLUSION

In this work, we successfully recreated the FLuID federated learning framework in a simulated setting without the need for physical devices. The recreation achieved comparable accuracy to the original FLuID paper, validating the simulation approach. Experiments benchmarking FLuID with the FedAdam optimization algorithm showed that while FedAdam provided a slight accuracy improvement over FedAvg, the increased training time makes the tradeoff less favorable, especially for resource-constrained environments. Additionally, this work proposed and evaluated two novel optimizations to further enhance FLuID's ability to mitigate straggler effects and communication bottlenecks: sparsity pruning and quantization. Sparsity pruning through raw and scaled thresholding techniques demonstrated promising accuracy while significantly reducing computational overhead compared to vanilla FLuID. However, the static nature of the thresholding values limited the full potential of this approach. Quantization, particularly with int8 precision, achieved substantial model size reduction at the cost of some accuracy degradation. This quantization tradeoff highlights the need to carefully balance model performance with computational and memory constraints for edge device deployments.

Overall, the findings from this work suggest several viable enhancements to the FLuID framework that can improve

its efficiency and scalability for federated learning scenarios with heterogeneous devices and limited resources. Future work could explore dynamic, adaptive sparsity and quantization techniques that better account for the evolving nature of the federated training process. Nonetheless, this project makes valuable contributions towards enabling practical, communication-efficient federated learning for a wider range of applications and resource-constrained environments.

7 COLLABORATION SUMMARY

Although all team members contributed evenly in training, testing, debugging, and designing this overall project, each team member was responsible for the delivery of a set of subcomponents in the overall development process. The report was distributed amongst members of the team evenly.

7.1 Arati Ganesh

Setup and recreation of FLuID framework on PACE ICE cluster. Implementation of quantization techniques. Evaluation of experimental results.

7.2 Aditya Iyer

Setup of FLuID extended with sparsity with Invariant Dropout. Evaluation of baselines and other experimental configurations.

7.3 Panchami Kamath

Implementation and integration of additional federated learning algorithms beyond FedAvg (FedAdam, FedOpt). Application of quantization. Evaluation of experimental results.

7.4 Faris Qadan

Implementation of additional federated learning algorithms beyond FedAvg (FedAdam, FedOpt) and potential extensions unaccounted for in the original work. Evaluation of experimental setups. Documentation of detailed findings into final poster.

REFERENCES

[1] Saar Barkai, Ido Hakimi, and Assaf Schuster. Gap aware mitigation of gradient staleness, 2020.

[2] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2022.

[3] Daniel Ramage Brendan McMahan. Federated Learning: Collaborative Machine Learning without Centralized Training Data. https://blog.research.google/2017/04/federated-learning-collaborative.html, 2017.

[4] Konstantin Burlachenko, Samuel Horváth, and Peter Richtárik. Fl_pytorch: optimization research simulator for federated learning. In *Proceedings of the 2nd ACM International Workshop on Distributed Machine Learning*, CoNEXT '21. ACM, December 2021.

[5] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konec ˇ ný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. 2018.

[6] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings, 2019.

[7] Sebastian Caldas, Jakub Konec ˇ ny, H. Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. 2018.

[8] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. Fedat: A high-performance and communication-efficient federated learning system with asynchronous tiers. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–17, 2021.

[9] Mingzhe Chen, Nir Shlezinger, H. Vincent Poor, Yonina C. Eldar, and Shuguang Cui. Communication-efficient federated learning. volume 118, page e2024789118, 2021.

[10] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24, 2020.

[11] Enmao Diao, Jie Ding, and Vahid Tarokh. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. In *9th International Conference on Learning Representations*, 2021.

[12] Francois Gauthier, Vinay Chakravarthi Gogineni, Stefan Werner, Yih-Fang Huang, and Anthony Kuh. Resource-aware asynchronous online federated learning for nonlinear regression. In *ICC 2022 - IEEE International Conference on Communications*, pages 2828–2833, 2022.

[13] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021.

[14] Ran Greidi and Kobi Cohen. Sparse training for federated learning with regularized error correction. *arXiv preprint arXiv:2312.13795*, 2023.

[15] Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. *Advances in Neural Information Processing Systems*, 33:14068–14080, 2020.

[16] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, 2021.

[17] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos I. Venieris, and Nicholas D. Lane. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout, 2022.

[18] Samuel Horváth, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Donald Lane. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. In *Advances in Neural Information Processing Systems*, 2021.

[19] Yuang Jiang, Shiqiang Wang, Bongjun Ko, Wei-Han Lee, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. 2022.

[20] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: stochastic controlled averaging for on-device federated learning. *CoRR*, abs/1910.06378, 2019.

[21] Alind Khare, Animesh Agrawal, Myungjin Lee, and Alexey Tumanov. Superfed: Weight shared federated learning. 2023.

[22] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity, 2020.

[23] Lumin Liu, Jun Zhang, Shenghui Song, and Khaled B. Letaief. Hierarchical federated learning with quantization: Convergence analysis and system design. *IEEE Transactions on Wireless Communications*, 22(1):2–18, January 2023.

[24] X. Ma, M. Qin, F. Sun, Z. Hou, K. Yuan, Y. Xu, Y. Wang, Y.-K. Chen, R. Jin, and Y. Xie. Effective model sparsification by scheduled grow-and-prune methods. In *International Conference on Learning Representations*, April 2022.

[25] Mohamed Hany Mahmoud, Abdullatif Albaseer, Mohamed Abdallah, and Naofal Al-Dhahir. Federated learning resource optimization and client selection for total energy minimization under outage, latency, and bandwidth constraints with partial or no csi. *IEEE Open Journal of the Communications Society*, 4:936–953, 2023.

[26] Yuzhu Mao, Zihao Zhao, Meilin Yang, Le Liang, Yang Liu, Wenbo Ding, Tian Lan, and Xiao-Ping Zhang. Safari: Sparsity enabled federated learning with limited and unreliable communications, 2022.

[27] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks, 2021.

[28] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, et al. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1):2383, 2018.

[29] Muhammad Tahir Munir, Muhammad Mustansar Saeed, Mahad Ali, Zafar Ayyub Qazi, and Ihsan Ayyub Qazi. Fedprune: Towards inclusive federated learning. 2021.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[31] Martin Rapp, Ramin Khalili, Kilian Pfeiffer, and Jörg Henkel. Distreal: Distributed resource-aware learning in heterogeneous systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8062–8071, 2022.

[32] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization, 2021.

[33] Amirhossein Reisizadeh, Isidoros Tziotis, Hamed Hassani, Aryan Mokhtari, and Ramtin Pedarsani. Straggler-resilient federated learning: Leveraging the interplay between statistical accuracy and system heterogeneity, 2020.

[34] Reent Schlegel, Siddhartha Kumar, Eirik Rosnes, and Alexandre Graell i Amat. Codedpaddedfl and codedsecagg: Straggler mitigation and secure aggregation in federated learning. 2021.

[35] Irene Wang, Prashant J. Nair, and Divya Mahajan. Fluid: Mitigating stragglers in federated learning using invariant dropout, 2023.

[36] Dingzhu Wen, Ki-Jun Jeon, and Kaibin Huang. Federated dropout—a simple approach for enabling federated learning on resource constrained devices. *IEEE Wireless Communications Letters*, 11(5):923–927, 2022.

[37] Di Wu, Rehmat Ullah, Paul Harvey, Peter Kilpatrick, Ivor Spence, and Blesson Varghese. Fedadapt: Adaptive offloading for iot devices in federated learning. 2021.