Towards cost-effective and resource-aware aggregation at Edge for Federated Learning

1st Ahmad Faraz Khan
Virginia Tech
Blacksburg, USA
ahmadfk@vt.edu

2nd Yuze Li
Virginia Tech
Blacksburg, USA
lyuze@vt.edu

3rd Xinran Wang
University of Minnesota
Minnesota, USA
wang8740@umn.edu

4th Sabaat Haroon
Virginia Tech
Blacksburg, USA
sabaat@vt.edu

5th Haider Ali
Virginia Tech
Blacksburg, USA
haiderali@vt.edu

6th Yue Cheng
University of Virginia
Charlottesville, USA
mrz7dp@virginia.edu

7th Ali R. Butt
Virginia Tech
Blacksburg, USA
butta@vt.edu

8th Ali Anwar
University of Minnesota
Minnesota, USA
aanwar@umn.edu

***Abstract*—**
Federated Learning (FL) is a machine learning approach that addresses privacy and data transfer costs by computing data at the source. It's particularly popular for Edge and IoT applications where the aggregator server of FL is in resource-capped edge data centers for reducing communication costs. Existing cloud-based aggregator solutions are resource-inefficient and expensive at the Edge, leading to low scalability and high latency. To address these challenges, this study compares prior and new aggregation methodologies under the changing demands of IoT and Edge applications. This work is the first to propose an adaptive FL aggregator at the Edge, enabling users to manage the cost and efficiency trade-off. An extensive comparative analysis demonstrates that the design improves scalability by up to $4\times$, time efficiency by $8\times$, and reduces costs by more than $2\times$ compared to extant cloud-based static methodologies.

***Index Terms*—federated learning, aggregation, edge computing**

I. Introduction

IoT and edge devices generate sensitive data that requires careful handling to prevent security or privacy violations. Traditional machine learning methods [1]–[3] that send private data to a centralized location for training pose significant privacy, communication, and security challenges and are unsuitable for sensitive data due to regulations like HIPAA [4] and GDPR [5]. Federated Learning (FL) [6] is a new technique that enables clients, such as IoT and edge devices, to collaborate in training a model without moving the data outside the system boundaries. FL reduces communication costs and privacy risks and has been adopted for a wide range of IoT and edge applications, including agriculture, healthcare, human behavior recognition, transportation, and smart homes [7]–[10].

Problem. At the heart of Federated Learning (FL) lies the parameter/aggregator server, which coordinates the training process, manages client participation, and performs aggregation of model updates. However, when it comes to IoT and edge applications, this server must contend with unique scalability, efficiency, and cost management challenges. This is especially true given the scale of these applications, which often involve millions of client devices with highly segregated data [11], [12]. As a result, addressing these challenges is critical to the success of FL in edge environments [13]. Edge data centers are compact data centers that have a smaller footprint compared to traditional data centers. They typically house only a limited number of servers, usually between 3 to 10, and can be easily moved and deployed in limited spaces such as a factory or an Internet Access point [14]. For this reason, it becomes an attractive option to house aggregator servers at the edge to save communication costs [9]. The intuition of unlimited resource scalability in existing solutions for cloud aggregator servers becomes an impractical approach at the edge data center. Limited compute, network, and memory resources at the edge data centers raise the requirement for a resource-aware scalable, and efficient aggregator at the edge while maintaining minimal costs.

Challenges. Existing studies on FL aggregation services have primarily focused on different aspects such as communication efficiency [15], time efficiency [11], [16], and cost reduction [17], [18]. However, these studies propose solutions for cloud settings where they typically assume no limitations on resources or costs for the aggregation server, therefore, these methodologies cannot be used for edge data center setups where resources are limited. Moreover, current FL frameworks [19]–[21] rely on a single central node for the aggregator server, leading to inefficiencies, high I/O or communication costs, prolonged aggregation times, and limited scalability. Furthermore, there is a lack of understanding of the complex system challenges that arise when developing

This work is sponsored in part by the NSF under the grants: CSR-2106634/2312785, CCF-1919113/1919075, OAC-2004751, and 3M.

scientific FL applications, especially in the context of edge-cloud aggregation jobs [22]–[24]. Thus, there is a need to develop new FL aggregation solutions that can effectively leverage edge data centers' resources while minimizing costs.

Contributions. In this paper, we aim to address the challenges of scalability, efficiency, and cost-effectiveness in FL aggregation at edge data centers. For this, we present a comprehensive analysis of the performance and cost of existing FL frameworks and multi-core, multi-node aggregation methodologies on limited resources, simulating conditions at the edge server. We acknowledge previous works such as serverless parameter servers [17], [18] and distributed aggregator servers [11], but find that they have limited scalability and efficiency or assume unlimited cost spending. Additionally, some designs rely on peer-to-peer communication, which increases communication costs and latency. To address these challenges, we propose a resource-aware adaptive aggregator design that selects the most efficient methodology while keeping resource spending low. Our design includes a Numba-based method, a Spark-based method [25], and a Serverless tree-reduce method, all integrated into a hybrid setting. This adaptive methodology enables the aggregator to scale according to demand while minimizing cost and latency. We use scalable storage for communication to overcome issues from peer-to-peer intra-aggregator connections. Additionally, users can customize the adaptive policy to achieve a balance between efficiency and cost trade-offs. In summary, this paper presents a novel resource-aware adaptive aggregator design for FL at edge data centers that address the challenges of scalability, time and resource efficiency, and cost-effectiveness. The goal is to reduce costs and spending on resources while maintaining the quality of service (QoS) indicated by low latency and cost-effective scalability for the user. To achieve this, the least costly method is chosen to meet the time efficiency and scalability requirements for particular aggregation jobs. Our approach differs from previous works by considering limited resources at the edge, and we believe this is the first work to do so.

Technical Insights. We start by building a predictive model that outputs a methodology for aggregation, minimizing time and monetary costs based on user preferences, using exhaustive profiling data, task-specific information, and system availability as input. To assess the efficacy of the adaptive method, we perform a comprehensive analysis of each technique by highlighting its strengths and weaknesses and comparing it with the adaptive method. We use a client emulator with a large number of clients and perform extensive micro and macro benchmarks to analyze each technique. Our analysis reveals interesting trends, such as serverless being more efficient than Spark for processing data from a large number of clients but costly for other specific workloads compared to Spark. On the other hand, Spark remains cost-effective and efficient for processing larger data chunks. Thus, each method has its pros and cons. No single method can fully address all the challenges in aggregation at the Edge data center, leading to the development of the adaptive aggregator design. Existing works have been done in the cloud setting, where the aggregator has unlimited resources. However, these methodologies cannot be used for edge data center setups with limited resources, a significant challenge our method addresses.

Evaluation. The evaluation consists of exhaustive experimentation with up to one million clients. Altogether, the duration of the experiments was approximately 1440 hours, and more than 100k lambda functions were used in total.

Summary. Our contributions are as follows:

1) Highlight the **limitations of existing FL frameworks** by analysis of common basic operations in their aggregation.
2) Propose and assess **three different strategies** for scalability, efficiency, and cost-effectiveness limitations.
3) Propose the **first solution for an adaptive aggregator at the Edge**. Our adaptive aggregation technique is driven by user preferences to achieve high QoS based on popular FL edge and IoT applications.
4) Provide a **client emulator** that connects to any cloud-based parameter server to conduct a cost-benefit analysis of static compared to the adaptive method.
5) Offer **Counter-intuitive insights**, like the unexpectedly high costs of Serverless for specific workloads driving the need for resource-efficient and cost-effective aggregation.

II. Background & Motivation

In the FL process, devices train local models and share them with an Aggregator (central) server. This Aggregator combines these local models, creating a global model using algorithms like Federated Averaging [6], Iterative Averaging, Gradient Aggregation, and Coordinate-wise median [26]. This global model goes back to clients for further training, repeating until the desired accuracy is reached. We use the IBM Federated Learning Library (IBMFL version 1.0.6) [20] as our baseline [20] referred to as Vanilla throughout the paper because it has a similar aggregator architecture as other common frameworks [1], [27]. **We only consider synchronous aggregation as it has gained popularity in recent works and is more stable in terms of convergence [13], [28]–[30]**. Next, we highlight challenges faced by the aggregator server of IoT and Edge FL applications such as edge data center's limited resources including computing, communication, and energy, along with varying participation and device damage [31], [32].

Varying clients' participation: In IoT and edge FL applications, client participation varies [9], [33], leading to varying model updates received by the aggregator. Conventional cloud servers cannot handle this in a resource-efficient manner as they are designed for unlimited scaling. An adaptive approach is needed to upscale for availability, and downscale for reducing cost, while maintaining efficiency. **Varying model sizes:** Techniques like model pruning [34], sparsification [35], and quantization [36] reduce communication costs for IoT. Models differ in size per client, posing a challenge for FL cloud solutions [1], [20], [27] lacking resource-aware scaling. Advanced methods using salient parameters [37] also cause varied model sizes per client device. Thus, an adaptive strategy is crucial to select the best aggregation method based on costs and resources. **Multi-tenancy Absence:** Edge cloud's

TABLE I: Specifications of models

Model	*Model Size*	*Convolutional layers*	*Dense layers*
CNN4.6	4.6 MB	32, 64	128
CNN73	73 MB	32, 256, 512, 1024	128
CNN179	179 MB	32, 512, 1024, 1900	128
CNN239	239 MB	32, 1024, 1900, 2400	128
CNN478	478 MB	32*2,1024*2, 1900*2, 2400*2	128*2
CNN717	717 MB	32*3, 1024*3, 1900*3, 2400*3	128*3
CNN956	956 MB	32*2,1024*2, 1900*2, 2400*2	128*4
Resnet50	91 MB	[44]	[44]
VGG16	528 MB	[45]	[45]

parameter and aggregator servers with multi-tenancy [38], [39] share limited resources at the edge data center. Existing FL aggregation techniques falter on Edge data centers because they assume unlimited resources for scaling. Thus, shifting to an efficient approach within confined resources is crucial.

Scaling for Dynamic Workloads: IoT and Edge FL applications depend on Edge data centers [7], [9], [10], [22], [40] for aggregation. These applications have inconsistent participation rates due to limited resources [41], and damage to client devices [31]. Furthermore, Aggregator servers are kept close to IoT devices in Edge datacenters to cut communication cost. However, they face challenges such as limited resources while serving multiple applications. Existing FL aggregator designs [20], [21], [27] lack the ability of resource-aware scaling for these variable workloads under limited resources. An adaptive aggregator is essential for flexible, resource-efficient scaling. **Improving Efficiency Performance:** Aggregation is performed after each epoch in the training process. Therefore, efficient aggregation can benefit all FL applications by reducing the waiting times of idle client devices [8], [10], [42] between training rounds, thus improving the quality of service (QoS) for clients and allowing them to utilize their resources for training. **User Autonomy for Controlling Costs:** Edge centers manage many applications [22], [43], and FL training is lengthy [13]. Saving energy and resources on edge aggregators is vital. A static approach might not scale well, costing more without downsizing.

We propose an adaptive aggregator design that dynamically adjusts its scale based on incoming workload, preventing overspending on resources and catering to workload surges. This flexibility enables users to balance between efficiency and cost, adapting to changing device and workload dynamics while prioritizing cost savings.

III. Methodologies and their Analysis

We present three diverse Federated Learning (FL) aggregation methods, leveraging technologies like Numba, Spark MapReduce, and Serverless with shared I/O channels, alongside implementation guidance and evaluations for limitations.

Focus of Analysis: We analyze the conditions favoring each method's performance to inform the design of an adaptive aggregation policy balancing time and cost efficiency. Through experimental analysis, we seek to address these key questions: ① What are the precise gains achieved by parallel fusion algorithm computation? (Section III-B) ② How does horizontal scaling impact time efficiency and cost in multi-node methodologies, and what is the upper limit on number of clients? (Section III-C) ③ How does the complexity of fusion algorithms affect memory and CPU bottlenecks in both multi-core and multi-node methods? (Sections III-B & III-C) ④ What bottlenecks emerge in horizontally scalable multi-node fusion algorithms? (Section III-C) ⑤ Which method suits specific use cases outlined in Section II, and what are their performance and cost advantages? (Section III)

A. Experimental Setup

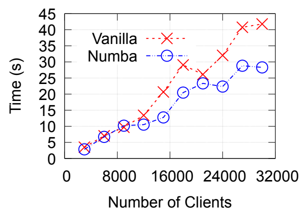
*1) **Models**:* We experimented with various CNN models, including Resnet50 [44] and VGG16 [45], in different sizes listed in Table I, systematically increasing model sizes for comprehensive analysis, with model sizes comparable to **MobileNetV2 (4.2 MB)**, **ShuffleNetV2 (70 MB)**, **ResNet50 (224 MB)**, **InceptionV3 (450 MB)**, and **VGG19 (518 MB)**.

*2) **Client Emulator**:* The client emulator streamlines client simulation for researchers by simplifying technical intricacies. It comprises three stages, utilizing AWS S3: first, it sends an HTTP request to AWS S3 to move files from client to aggregator buckets, adjusting to regions and model quantities. Then, it uses monitoring to wait for the aggregated model in the aggregator's bucket. Finally, it triggers multiple S3 requests to return the model to the clients' bucket. Furthermore, the client emulator allows the simulation of user-defined randomized dropouts.

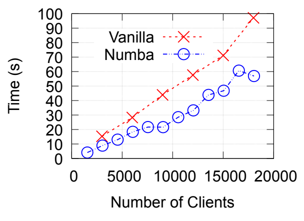
*3) **Testbed**:* **Spark-based Methodology:** We evaluated the Spark-based method using a Spark cluster with 256 cores and 452 GB of memory. Apache Spark [25] version 3.2.0 ran on Apache Hadoop Yarn [46] version 3.2.2. Executors in Spark had a 35 GB memory limit. We used HDFS with 2.6 TB storage for model updates, and executor container specifications were adjusted based on workload. **Serverless Methodology:** For the Serverless method, we employed AWS Lambda with a 4 GB memory limit per function. **Numba-based Methodology:** To compare the Numba-based method, we used a containerized setup on a single node with 64 cores, 256 GB memory, and 10 Gbit/s network bandwidth. The client emulator from Section III-A2 was used, with a five percent client sample in all experiments. Our analysis design was implemented in around 2K lines of Python code.

*4) **Metrics**:* We assessed four key metrics to evaluate the efficiency of each methodology: ① Client sampling rate for aggregation per round (default at 5% of total clients), indicating scale achieved. ② Time for aggregation, assessing its efficiency. ③ Aggregation cost, reflecting the value for money spent. ④ Communication time, representing latency in model updates between clients. We also measured client write and read efficiency from the scalable storage used by the aggregator in our client emulator. These metrics collectively offer insights into the resource efficiency of each approach.

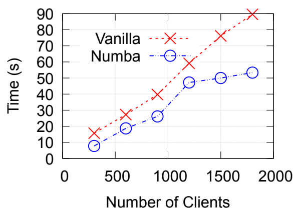
*5) **Fusion Algorithms**:* We analyzed two fundamental averaging-based fusion algorithms: Federated averaging (FedAvg) and Iterative averaging (IterAvg) from Vanilla. These serve as the basis for various fusion methods, including ClippedAveraging, ConditionalThresholdAveraging (discussed



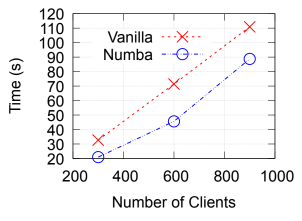
(a) Iteravg (CNN4.6)



(b) Fedavg (CNN4.6)



(c) Iteravg (Resnet50)



(d) Fedavg (Resnet50)

Fig. 1: Average aggregation time comparison of Vanilla and Numba for smaller (CNN4.6) model and Resnet50 model

in [19]), and Gradient Aggregation algorithms (mentioned in [20]). FedAvg, defined in Equation 1, employs client weights (w_i) with a total client count (n_{total}) and a small ϵ value of 10^{-6}. Additionally, we are exploring more advanced fusion techniques like Zeno [47] and Coordinate-wise median [26].

$$M = \sum_{i=1}^{n} w_i/(n_{total} + \epsilon) \tag{1}$$

B. Multi-core Aggregation

Our FL aggregation was optimized by switching from Numpy [48] to Numba [49], enhancing parallel processing. This change, implemented in IBMFL's FedAvg and IterAvg classes, retained the original communication and client training functions. It enabled storing client updates in the aggregator's memory for quicker, more efficient aggregation, especially beneficial for IoT applications with simpler models. This transition to Numba required only minor code modifications within the FL framework.

To answer question ① in section II, we created microbenchmarks to analyze the Numba-based aggregation method compared to the Vanilla implementations of FedAvg and IterAvg algorithms. Figures 1c and 1d show the aggregation time comparison for the Resnet50 model. For Fedavg, we observed a 40.44% reduction in execution time using the Numba-based method with 1.8k clients. However, due to fixed single-node memory, the number of clients supported by Resnet50 is lower compared to a smaller model like CNN4.6, which is why the Numba-based method performs similarly to Vanilla for the Resnet50 model. With higher participation, the Numba-based method can parallelize computations and improve time efficiency. This is evident in Figures 1a and 1b for the CNN4.6 model, where the Numba-based method reduced aggregation time by 56.85%. Moreover, the efficiency of the Numba-based method depends on the fusion algorithm used. Numba parallelizes loops to compute weighted averages in Fedavg, resulting in greater efficiency, while simpler calculation in Iteravg means fewer efficiency gains from parallel computation.

In summary, the Numba-based method outperforms IBMFL and other FL frameworks that use the Numpy library for implementing fusion algorithms when client participation rates increase. It is also more resource-efficient as it utilizes all available cores to maintain efficiency reducing resource idling. However, for a smaller number of clients, parallel processing is less beneficial. These insights can be extended to other frameworks such as TensorFlow [1], Microsoft FLUTE [27], and FLOWER [50]. This analysis also suggests that multiple resources (CPU, Memory) act as bottlenecks during aggregation. We also note that when we move from IterAvg to FedAvg, which is a slightly more complex algorithm, the CPU becomes a bottleneck. We conclude that the **CPU bottleneck is directly related to the complexity of the algorithm**, and as we move to more complex algorithms, the bottleneck induced by CPU resources will increase, answering question ③ raised in section II. Considering this result, we argue that **specialized hardware (GPU)** as a solution cannot fully resolve multiple bottlenecks and can be expensive. This motivates us to explore and analyze further aggregation methodologies that can be used for larger workloads using CPU.

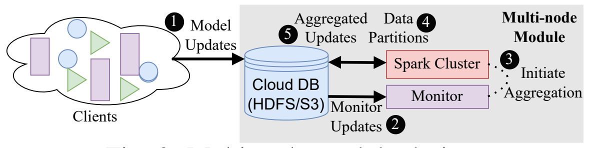
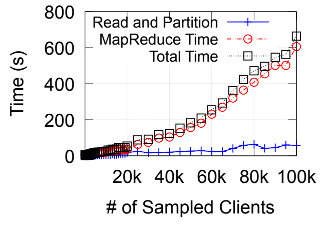
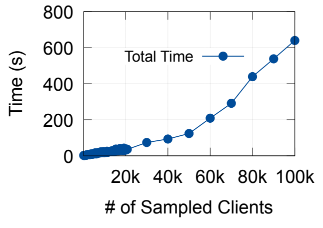


Fig. 2: Multi-node module design



(a) Fedavg



(b) Iteravg

Fig. 3: Average aggregation time comparison of the Spark method with Fedavg and Iteravg on the CNN4.6 model

C. Spark-based Multi-node Aggregation

The Spark-based aggregator in the adaptive aggregation methodology, shown in Figure 2, includes a scalable storage monitor. This monitor activates Spark for aggregation after a timeout or a set number of client updates. The threshold is adjustable to mitigate stragglers. This lightweight monitor runs efficiently as a single-threaded process.

The Spark module performs aggregation in the following steps: ❶ After each training round, client-sent model updates are stored in Hadoop Distributed File System (HDFS) using the webHDFS Rest API. ❷ The monitor waits for a threshold to be reached and triggers the Spark cluster module to initiate aggregation. ❸ Spark partitions the data and employs the binary files method to read data as bytes in executor containers. The map function converts RDD bytes into RDDs of the Numpy object type. ❹ Lastly, Spark MapReduce processes

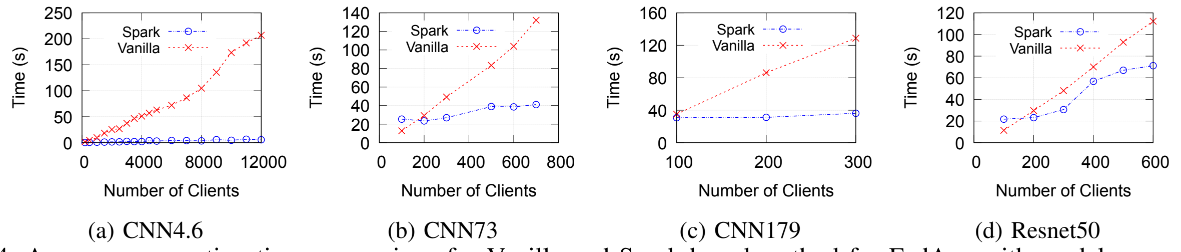

(a) CNN4.6 (b) CNN73 (c) CNN179 (d) Resnet50

Fig. 4: Average aggregation time comparison for Vanilla and Spark-based method for FedAvg with model compression

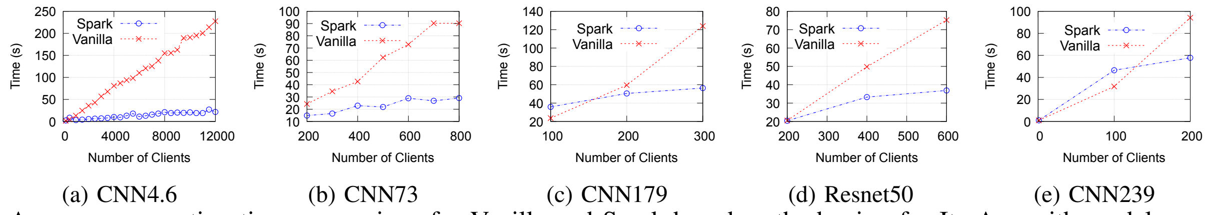

(a) CNN4.6 (b) CNN73 (c) CNN179 (d) Resnet50 (e) CNN239

Fig. 5: Average aggregation time comparison for Vanilla and Spark-based method using for IterAvg with model compression

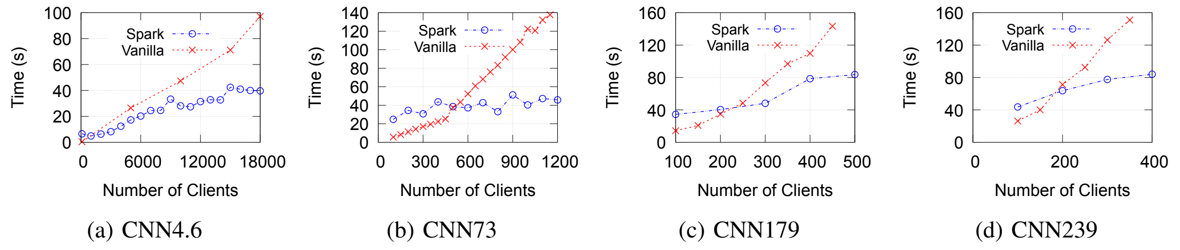

(a) CNN4.6 (b) CNN73 (c) CNN179 (d) CNN239

Fig. 6: Average aggregation time comparison for Vanilla and Spark-based method for FedAvg without model compression

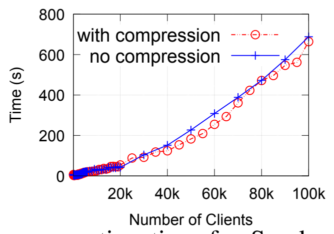


Fig. 7: Average aggregation time for Spark with FedAvg on CNN4.6 model with and without model compression

RDDs, applies the fusion algorithm, and stores updated model weights in HDFS for client access via the WebHDFS API.

In a detailed experiment, we tested the Spark-based method with an increasing number of clients using the FedAvg algorithm and the CNN4.6 model. The results in Figure 3a show ”MapReduce Time” as the time for weighted averaging of partitions by this method. We used a ”lazy read” approach for efficiency, reading only the partition in use. Smaller models were cached, with most RDDs of model weights cached on worker nodes until the reduction step. However, caching was less effective for larger models due to memory constraints.

IoT and Edge devices have limited network resources [31], [41]. Compression techniques can affect computation cost, scalability, and aggregator efficiency. In Figure 4, multi-node approaches prove more effective for aggregating compressed models in IoT and Edge FL, reducing compute cost, network usage, and aggregation time for clients.

Figure 3 displays the total time required for the Iteravg method, which involves only two simple steps of sum and division for a mean calculation. The number of clients selected per training round was increased iteratively up to 100k in Figure 3. Both FedAvg and Iteravg displayed a linear trend in the time required to aggregate, and no limitations were observed in their ability to scale horizontally using this Spark-based method. The number of clients supported per training round increased by 429.1% for FedAvg and 207.7% for Iteravg compared to Vanilla. It is important to note that the figure only shows up to 100k clients, but the adaptive method which includes the Spark-based method has the potential to scale for even more clients and 100k represents only 5% of the clients that are selected from the total available clients per round for aggregation which means the total clients can be as much as 2 million. This analysis provides an answer to the scalability question ② (raised in section II). Although Spark showed scalability, a detailed analysis of its resource efficiency and costs is necessary to examine its practicality for the edge aggregator, which is performed in section IV.

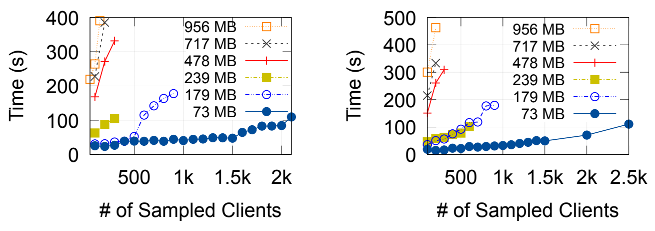

(a) Fedavg (b) Iteravg

Fig. 8: Average aggregation time for Fedavg and Iteravg algorithms on varying model sizes with Spark

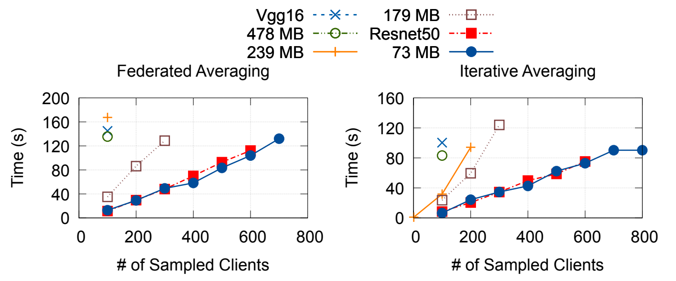


Fig. 9: Average aggregation time of Vanilla with varying models & compression under 170 GB total available memory

We also evaluated various benchmark models that use compressed model updates, which is a technique commonly used to reduce communication costs at edge data centers and IoT devices with limited network bandwidth [41]. The evaluation results are presented in Figures 4 and 5. The Spark-based method was found to be the most time-efficient when there was a large number of clients, as shown in Figures 4a and 5a. This is because Spark enables parallel tasks to be executed across multiple executors, thereby increasing the efficiency with higher client participation. To ensure unbiased results, we also conducted the same evaluation without compression. The results, as illustrated in Figure 6, exhibit a similar trend. However, the time efficiency gain is more significant compared to Vanilla with compression in Figure 4.

In the Spark-based method, the workload is split into multiple workers with separate system resources, reducing the latency involved in the decompression phase as each worker only has to decompress a portion of the total compressed workload. This is more clearly demonstrated in Figure 7, where the aggregation latency is similar with or without compressed model updates. In the Vanilla method, decompression with multi-threading is still slow due to the limited memory and CPU resources. This is further evidenced by the Vanilla latency difference between Figures 4a and 6a.

In summary, while compression increases latency and memory use, thus limiting scalability with Vanilla and Numba methods, it boosts efficiency and scalability with Spark due to its parallel task handling and workload distribution abilities. This makes the Spark-based method more resource-efficient at scale, reducing communication costs at the edge aggregator and maintaining time efficiency.

Figure 6 shows that Spark works better for heavy models and high client participation surpassing the I/O cost, while Numba is more efficient and cost-effective for lighter models and low participation rates (Sections III and IV-A2). Thus, an adaptive aggregator is crucial to choose the right method based on IoT and Edge workload and participation rates, ensuring cost-effectiveness.

The analysis is extended with compression for different models and increasing numbers of clients in Figure 8. For Fedavg, the Spark-based method shows a 3X increase in scalability compared to Vanilla. For the CNN73 model, the Spark-based method shown in Figure 8a can scale to more than 2.1k clients while Vanilla in Figure 9 only scales to 700 clients under the same evaluation settings described in section III-A3. Similarly, IterAvg in Figure 8b also shows a 3X improvement in scalability compared to Vanilla in Figure 9.

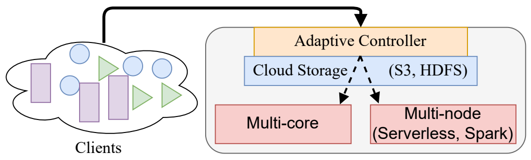


Fig. 10: Adaptive Aggregator Design

D. Serverless-based Multi-node Aggregation

The Serverless method uses the tree-reduce principle and the monitor shown in Figure 2 to control the multi-level reduction of updates. At each reduction step, the monitor launches multiple Lambda functions concurrently, with the number of updates handled per function dynamically calculated to ensure the memory per function is a maximum of 4 GB. More functions are launched for horizontal scaling to handle additional clients. The Lambda functions take input from the scalable storage used by the Spark module, and the intermediate and final updates are stored in the same scalable storage. In step ❸, the monitor sends AWS Simple Notification Service (SNS) messages to launch Lambda functions for aggregation. Users can adjust the number of Lambda functions and memory limits for each function to optimize efficiency and cost.

Tree-reduce algorithm efficiently distributes workloads in multi-node settings, as demonstrated in Figures 4a, 5a, and 6a. This makes multi-core and multi-node aggregator architectures more suitable for Edge and IoT FL applications than Vanilla and other frameworks due to their ability to process workloads in parallel.

For complex workloads in IoT and edge FL applications, Spark excels in efficiency over Serverless, while Serverless is more economical and quicker for simpler models due to reduced per-function costs. Thus, in general, Spark is better for heavier models [51], and Serverless for lighter ones [7].

Experimental analysis shows in Figures 11a and 11b that the Serverless method is more efficient and less costly than the Spark-based method for lighter models, but more costly than the Numba-based method when the participation rate of clients is low. Compared to Spark, the Serverless method improves the aggregation efficiency by up to 87% for both FedAvg and IterAvg. This analysis emphasizes the need for an adaptive aggregator, as the Serverless method becomes cost-effective for lighter workloads and can handle unpredictable participation rates of IoT devices, but becomes more costly when the participation rates are lower or the aggregator is dealing with heavier models. The next section provides a more detailed analysis of the Serverless method.

IV. Adaptive Aggregator

This section presents the design of the adaptive aggregator given in Figure 10, which uses insights from the analysis in section III to develop a predictive model for estimating

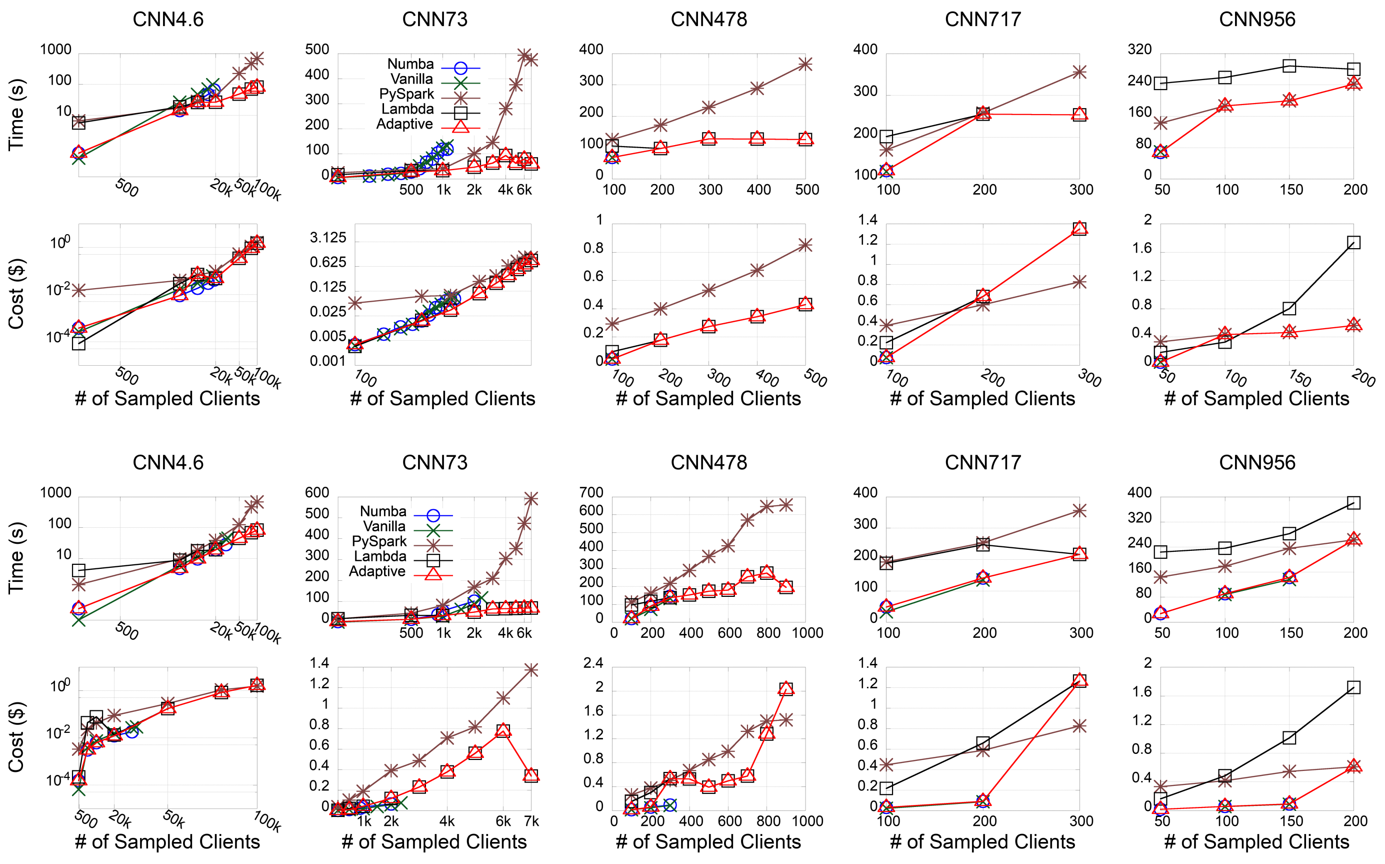


Fig. 11: Comparison of different methodologies for FedAvg (top) and IterAvg (bottom) without compression.

the completion time of aggregation tasks for each available method. We also conduct an exhaustive analysis of the adaptive method and demonstrate its advantages through a cost, scalability, and efficiency comparison with other methodologies.

We employ a Q-learning-based Reinforcement Learning (RL) agent to optimize the tradeoff between cost, resource utilization, and time efficiency. Let Y_i represent the time taken for a user's aggregation task, which depends on multiple state variables: task-specific information (S_T), and system resource data (S_R). Task-specific information includes workload, calculated as the product of the number of clients and the size of model parameters. System information includes available memory, CPU capacity, and the number of executor containers/functions for Spark or Serverless computing. The RL agent learns from these state variables and selects an aggregation method (A) to optimize the tradeoff between cost and completion time (Y_i), refining its understanding through interactions with the environment. Here's an overview of the algorithm:

Algorithm 1 optimizes aggregation selection through RL. It initializes Q-values (Line 2), explores or exploits based on Q-values (Lines 3-9), executes aggregation methods (Line 10), observes completion time and calculates rewards based on user preferences (Lines 11-16), and updates Q-values (Line 17). Ultimately, the method with the highest Q-value is chosen for efficiency (Line 19). Q-value is updated for the chosen action using the Q-learning update equation, incorporating the observed reward, learning rate (γ), and discount factor (μ). The reward calculation takes into account the user's preference for either time efficiency or cost-effectiveness. If T/C is True, it implies the user prefers time efficiency, so the reward is based on negative completion time (-Y_i). If T/C is False, indicating the user prefers cost-effectiveness, the reward is calculated by subtracting the cost of aggregation and adding a penalty based on completion time (P_c * Y_i). This encourages the agent to minimize completion time while taking into account the user's preference for time efficiency and cost-effectiveness. Hyperparameters ϵ, γ, and μ are fine-tuned through sensitivity analysis and experimentation to achieve the desired balances between exploration and exploitation, learning rate, and discount factor. The adaptive aggregator RL agent optimally adjusts aggregation methods based on Q-values and observed state variables, ensuring efficient tradeoffs between cost, resource use, and time, outperforming other methods in scalability, efficiency, and cost-effectiveness.

A. *Cost-benefit Study with Adaptive Method*

The adaptive aggregator can customize cost, scalability, and efficiency requirements for FL aggregators at the edge, making it the first adaptive FL aggregator for Edge and IoT applications. By default, the system chooses the most resource-efficient method based on cost, which is directly translated from resource consumption. This is done while

Algorithm 1: RL-based Adaptive Aggregator

```
1  Input: $S_T$: Task-specific information, $S_R$: System
     resources information, $\epsilon$: Exploration probability, $\gamma$:
     Learning rate, $\mu$: Discount factor, $N$: Number of
     epochs, $T/C$: User preference for time efficiency or
     cost-effectiveness, if True user prefers efficiency, $P_c$:
     Penalty factor for completion time, $C_a$: Cost of
     aggregation
2  Initialize Q-values $Q(S_T, S_R, A)$ with random values
3  for $epoch = 1$ to $N$ do
4      Observe state variables $S_{uc}$, $S_T$, $S_R$
5      if random value $< \epsilon$ then
6          Select $A$ randomly for exploration
7      else
8          Choose $A$ with highest $Q$-value for exploitation
9      end
10     Execute aggregation method $A$
11     Observe completion time $Y_i$
12     if $T/C$ is True then
13         $R = -Y_i$
14     else
15         $R = -C_a + P_c \cdot Y_i$
16     end
17     Update Q-value: $Q(S_T, S_R, A) \leftarrow (1-\gamma) \cdot$
         $Q(S_T, S_R, A) + \gamma \cdot (R + \mu \cdot \max_{A'} Q(S_T, S_R, A'))$
18 end
19 Return aggregation method $A$ with highest Q-value:
     $A = \arg\max_A Q(S_T, S_R, A)$
```

ensuring that the QoS requirements, including time efficiency and scalability, are maintained for the user.

1) Cost Calculation: The Serverless method's cost is determined by AWS Lambda and depends on function memory and billing duration. Storage costs (S3) are variable and depend on storage characteristics. For the Spark-based method, cost calculation is possible using the pay-to-use AWS Glue API, supporting Serverless ETL operations with PySpark [52].

2) Multi-core in Adaptive Aggregator: In fewer-participant IoT applications, both FedAvg and IterAvg start strong with Numba, but face efficiency drops from memory-induced CPU bottlenecks as memory limits are reached, regardless of fusion algorithm complexity, as shown in Figure 11 with log scales.

Figure 11 shows that a fixed multi-node method is overkill for less complex models with low participation rates which makes it more expensive and less efficient than Numba.

3) Multi-Node in Adaptive Aggregator: Algorithm 1 provides an optimal performance in time and cost, as Figure 11 illustrates. This adaptive aggregator switches to a Serverless method when client numbers surpass 15k for various CNN models, resulting in significant time and cost savings. For CNN4.6, it reduces latency by 88% and for CNN73 by 85.59%, with a \$0.2 cost saving per run compared to a static Spark method. With CNN478, latency decreases by 63.51%, while for heavier models like CNN956, where Serverless costs

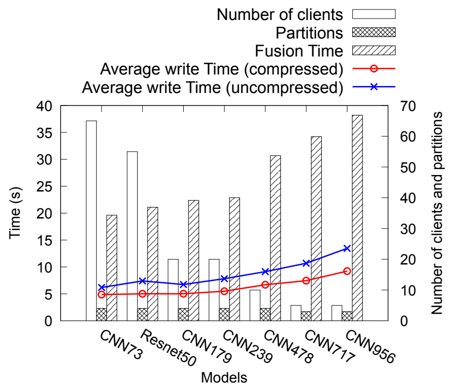


Fig. 12: An end-to-end working comparison of the Spark method with simulated clients for varying models using FedAvg (y-axis on the right shows the number of clients and Partitions, y-axis on the left represents all the bars)

increase due to higher cold-start container requirements, the aggregator opts for Spark-based methods, increasing efficiency by 28.73% and cutting costs by \$1. Although with CNN717 the adaptive method's costs are higher at greater participation levels, favoring efficiency, it can be adjusted to prefer the less expensive Spark-based method, saving \$0.6 each round.

The key point from the analysis in Section III is that an adaptive approach is best for addressing diverse workloads in various FL applications. This approach can strike a balance between enhancing efficiency and scalability while also lowering costs through resource-efficient decision-making.

Similar patterns are observed with the IterAvg algorithm in Figure 11, where the adaptive method yields savings in both time and cost. Specifically for workloads like CNN956, the adaptive approach reduces time by 118s and saves over 1 dollar each training round, substantial savings over the hundreds to thousands of rounds in a single FL training job. This algorithm dynamically switches between Numba, Serverless-based, and Spark-based methods, achieving notable cost reductions per round and enhancing QoS, as summarized in Table II.

Thundering herd problem can occur if all clients send their data at the same time [11], [13], especially with millions of IoT or edge devices with unpredictable participation rates [42], [53], but this effect can be mitigated by using scalable storage (S3/HDFS) with multi-node aggregation.

Communication in Adaptive Aggregation: We assessed the adaptive method's end-to-end latency in small-sized edge data centers [24], [41], focusing on the Spark-based method within the adaptive aggregator. Our experiment involved simulated clients on 6 machines connected via a 1 Gigabit Ethernet switch and used scalable HDFS storage for communication (details in section III-A3). To avoid client-side network bottlenecks, we adjusted the number of simulated clients based on network and machine capacities for various model sizes, ensuring the aggregator's write throughput was tested without network constraints on the client side. The results in Figure 12 reveal that even with model update sizes increasing by over

TABLE II: Total aggregation time comparison of different aggregation methodologies with the Adaptive method

	Supported Model Size	Scalability	Latency	Average Cost
Single Node	Very Small	Does not scale	Low	$
Serverless-TreeReduce	Medium	High (except large models)	Medium	$$
Spark	Large	High	High	$$$
Adaptive	All	High	Method dependent	$$

TABLE III: Aggregation with multiple tenants

Methodologies	Models	# of Sampled Clients	Cost ($)	Total Time (s)
Serverless Tree-Reduce	Resnet50	900	0.088	50.05
	Vgg16	100	0.153	117.73
Spark MapReduce	Resnet50	900	0.217	93.60
	Vgg16	100	0.307	132.36
Numba	Resnet50	400	0.019	29.94
	Vgg16	80	0.038	59.74

9X, the write time only slightly varied. The figure also presents the number of clients, Spark's task partitions for each model aggregation, and the time for reading, writing, and fusion. The average write time refers to the time for writing a single model update from a client, while the reduced time pertains to the MapReduce time for computing the weighted average of partitioned data. We maintained constant partition numbers (Spark tasks) to observe the impact of model size increase on fusion time. Thus, the adaptive aggregator efficiently utilizes scalable storage for client I/O and data input to aggregation task executors (Serverless functions/Spark executors).

B. Supplementary Features

TABLE IV: Emulator write throughput

No drop out				Drop out 5%			
# Clients	**Client Locations**	**CNN4.6 (s)**	**CNN478 (s)**	**# Clients**	**Client Locations**	**CNN4.6 (s)**	**CNN478 (s)**
1000	Ireland	1.99	109.77	1000	Ireland	1.64	97.42
	Seoul	3.401	120.84		Seoul	2.55	97.58
	California	1.84	98		California	1.66	99.65
	Total Time	13.98	238.19		Total Time	3.69	116.29
50000	Ireland	1.65	114.27	50000	Ireland	1.8	113.83
	Seoul	5.441	126.76		Seoul	5.15	125.79
	California	1.59	116.68		California	1.73	111.64
	Total Time	114.48	12195.45		Total Time	84.42	11670.50

Multi-tenant Isolation: The edge data center's adaptive aggregation service supports multi-tenancy and was tested with VGG16 and Resnet50 models. The results in Table III show that Serverless and Spark performed well with many clients, while the Numba-based method had a limit of 400 clients due to memory constraints. Resource management for multiple tenants is a well-researched topic in cloud services [38], [39], but it's outside our paper's scope. **Emulator Evaluation:** Table IV summarizes our emulator evaluation. We tested clients in three global regions, using both small (1000) and large (50k) client numbers, with simpler (CNN4.6) and more complex (CNN478) models. The aggregator server was in Virginia, USA. We measured average client write times and total write times in seconds. Closer locations to the aggregator server (e.g., California) had lower latency, while distant ones (e.g., Seoul) had higher latency due to longer network distances. We also simulated dropouts, randomly dropping 5% of clients as stragglers, which reduced latency, improving performance. This emulator is valuable for assessing parameter server and FL aggregator performance. **Other Multi-node Methods:** We also did an experimental evaluation with Dask, however, it performed less efficiently than Spark due to spending more time on I/O and conversion to its native Bag type. Spark offers better read-and-write throughput with cloud storage and efficiently partitions data for MapReduce computations. **Seamless Transition:** The adaptive aggregation service seamlessly switches methodologies for different workload sizes to avoid disrupting clients during FL. The I/O channel remains the same in all methods, facilitating smooth transitions. We use the WebHDFS Rest API for HDFS transfers and the Boto3 AWS SDK for Python for S3 storage and retrieval. Clients employ the same APIs for updates. Serverless startup costs are minimal, while Spark context startup costs are hidden during training. **Deployment to the Edge:** The adaptive aggregator can be deployed at the edge using services like AWS Lambda@Edge [54], AzureIoTEdge [55], and OpenEdge [56] for serverless deployment. Numba can run on a simple Linux container, and Spark on Linux-based nodes. **Convergence Guarantees:** In terms of convergence guarantees, the adaptive aggregator ensures the same level of convergence as other systems, as it uses the same fusion algorithm and formula. The difference lies only in the computation technique, without affecting the number of training rounds or final accuracy.

V. Related Work

Heterogeneity Aware FL: Lai et al. [12] proposes Oort, a new participation selection scheme for FL clients. This work looks at improving efficiency from the client's side. It evaluates up to 1.3k clients with small-sized models, out of which 100 are selected by default for aggregation which is approximately 1000X less than the scale we demonstrate. *Hierarchical Aggregation:* Bonawitz et al. [11] suggest selecting a smaller ratio from available clients to train and creating a hierarchy in the aggregator for scaling but this increases the number of rounds to converge. Liu et al. [24] suggest a hierarchical FL system in which partial aggregation is done at the edge to distribute load but does not consider the fault tolerance or robustness in the edge and cloud aggregators. Having such geographically distributed partial aggregators increases communication time between aggregators and adds extra I/O at each partial aggregator. This work only claims to support up to 1000 lightweight clients in cross-device settings and only a few heavier model clients in cross-silo settings. *Serverless Aggregation:* Grafberger et al. [17] propose a serverless solution for both the client and the server. Due to the short-lived nature of Lambda functions it needs to create special provisions to support larger model training and does not mention aggregator scalability. Jayaram et al. [18] suggest a serverless aggregator which is horizontally scalable with a Kubernetes [57] cluster. This reduces the cost of aggregation by using the pay-to-execute model with serverless functions but has the same limitations as [17]. [58] proposes PAPAYA: An asynchronous FL system, however in this paper our main focus is on synchronous FL solutions. Almost all the fore-mentioned methods assume that there are unlimited network, computation, and memory resources at the Edge, which is unrealistic. In addition, these static techniques are all cloud-based solutions that cannot handle each workload with cost-effectiveness and resource-efficiency, leading to a degradation in the QoS for the user. Furthermore, none of the works tackle all three challenges of scalability, efficiency, and cost reduction.

VI. Conclusion

FL is increasingly used in Edge and IoT with edge data center servers to cut communication costs. However, conventional cloud-based aggregators, designed for unlimited resources, face challenges with scalability and efficiency, resulting in higher latency and costs. This study introduces an adaptive aggregator that selects from three methodologies to improve scalability and resource efficiency, and to reduce costs and latency. This adaptive approach also provides users control over cost and efficiency, offering insights into FL aggregation's challenges in edge data centers for IoT and Edge applications, emphasizing the need for flexible solutions.

References

[1] M. Abadi *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, 2016.

[2] T. Chilimbi *et al.*, "Project adam: Building an efficient and scalable deep learning training system," in *OSDI*, 2014.

[3] H. Ali *et al.*, "Acadia: Efficient and robust adversarial attacks against deep reinforcement learning," in *IEEE CNS*, 2022.

[4] A. Act, "Health insurance portability and accountability act of 1996," *Public law*, vol. 104, p. 191, 1996.

[5] G. D. P. Regulation, "General data protection regulation (gdpr)," *Intersoft Consulting, Accessed in October*, vol. 24, no. 1, 2018.

[6] B. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017.

[7] A. Maroli *et al.*, "Applications of iot for achieving sustainability in agricultural sector: A comprehensive review," *Journal of Environmental Management*, vol. 298, p. 113488, 2021.

[8] K. S. Arikumar *et al.*, "Fl-pmi: Federated learning-based person movement identification through wearable devices in smart healthcare systems," *Sensors*, vol. 22, 2022.

[9] M. Abdel-Basset *et al.*, "Federated intrusion detection in blockchain-based smart transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, pp. 2523–2537, 2022.

[10] T. Zhang *et al.*, "Federated learning for the internet of things: Applications, challenges, and opportunities," *IEEE Internet of Things Magazine*, pp. 24–29, 2022.

[11] K. A. Bonawitz *et al.*, "Towards federated learning at scale: System design," in *SysML 2019*, 2019.

[12] F. Lai *et al.*, "Oort: Efficient federated learning via guided participant selection," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 19–35.

[13] P. Kairouz *et al.*, "Advances and open problems in federated learning," 2019. [Online]. Available: https://arxiv.org/abs/1912.04977

[14] Gartner *et al.*, "Predicts 2022: The distributed enterprise drives computing to the edge," www.gartner.com/document/4007176, 2022.

[15] J. Kone *et al.*, "Federated learning: Strategies for improving communication efficiency," *CoRR*, 2016.

[16] M. Ekmefjord *et al.*, "Scalable federated machine learning with fedn," *arXiv*, 2021.

[17] A. Grafberger *et al.*, "Fedless: Secure and scalable federated learning using serverless computing," in *IEEE Big Data*, 2021, pp. 164–173.

[18] K. R. Jayaram *et al.*, "λ-fl : Serverless aggregation for federated learning," 2022.

[19] G. A. Reina *et al.*, "Openfl: An open-source framework for federated learning," 2021.

[20] H. Ludwig *et al.*, "Ibm federated learning: an enterprise framework white paper v0. 1," 2020.

[21] M. Abadi, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: www.tensorflow.org/

[22] G. Premsankar *et al.*, "Edge computing for the internet of things: A case study," *IEEE IoT-J*, pp. 1275–1284, 2018.

[23] I. Thangakrishnan *et al.*, "Herring: Rethinking the parameter server at scale for the cloud," in *SC*, 2020.

[24] L. Liu *et al.*, "Client-edge-cloud hierarchical federated learning," in *ICC*, 2020, pp. 1–6.

[25] M. Zaharia *et al.*, "Spark: Cluster computing with working sets," in *USENIX Conference on Hot Topics in Cloud Computing*, 2010.

[26] D. Yin *et al.*, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *ICML*, 2018.

[27] D. Dimitriadis *et al.*, "Flute: A scalable, extensible framework for high-performance federated learning simulations," 2022.

[28] M. Aledhari *et al.*, "Federated learning: A survey on enabling technologies, protocols, and applications," *IEEE Access*, 2020.

[29] A. F. Khan *et al.*, "Pi-fl: Personalized and incentivized federated learning," *arXiv*, 2023.

[30] J. Han *et al.*, "Tiff: Tokenized incentive for federated learning," in *IEEE CLOUD*, 2022, pp. 407–416.

[31] M. M. Hossain *et al.*, "Towards an analysis of security issues, challenges, and open problems in the internet of things," in *2015 IEEE World Congress on Services*, 2015.

[32] H. Ali *et al.*, "A survey on attacks and their countermeasures in deep learning: Applications in deep neural networks, federated, transfer, and deep reinforcement learning," *IEEE Access*, vol. 11, 2023.

[33] J. Han *et al.*, "Heterogeneity-aware adaptive federated learning scheduling," in *IEEE Big Data*, 2022, pp. 911–920.

[34] Y. Jiang *et al.*, "Model pruning enables efficient federated learning on edge devices," *IEEE TNLS*, 2022.

[35] A. Albasyoni *et al.*, "Optimal gradient compression for distributed and federated learning," *CoRR*, vol. abs/2010.03246, 2020.

[36] S. Zheng *et al.*, "Design and analysis of uplink and downlink communications for federated learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, pp. 2150–2167, 2021.

[37] S. Yu *et al.*, "Spatl: Salient parameter aggregation and transfer learning for heterogeneous clients in federated learning," 2021.

[38] J. Mace *et al.*, "2dfq: Two-dimensional fair queuing for multi-tenant cloud services," in *SIGCOMM*, 2016.

[39] B. P. Rimal and M. Maier, "Workflow scheduling in multi-tenant cloud computing environments," *IEEE TPDS*, 2017.

[40] J. Pan and J. McElhannon, "Future edge cloud and edge computing for internet of things applications," *IEEE IoT-J*, 2018 pages=439-449,.

[41] S. Dey *et al.*, "Challenges of using edge devices in iot computation grids," in *ICPADS*, 2013.

[42] J. Kang *et al.*, "Reliable federated learning for mobile networks," *IEEE Wireless Communications*, 2020.

[43] S. Khalid and C. Brown, "Software engineering approaches adopted by blockchain developers," in *IEEE SDS*, 2023.

[44] K. He *et al.*, "Deep residual learning for image recognition," 2015.

[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[46] V. K. Vavilapalli *et al.*, "Apache hadoop yarn: yet another resource negotiator," *SOCC*, 2013.

[47] C. Xie *et al.*, "Zeno: Byzantine-suspicious stochastic gradient descent," *CoRR*, vol. abs/1805.10032, 2018.

[48] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, 2020.

[49] S. K. Lam *et al.*, "Numba: A llvm-based python jit compiler," in *Workshop on the LLVM Compiler Infrastructure in HPC*, 2015.

[50] D. J. Beutel *et al.*, "Flower: A friendly federated learning research framework," 2022.

[51] Y. Niu *et al.*, "Federated learning of large models at the edge via principal sub-model training," in *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NIPS)*, 2022.

[52] K. Sudhakar, "Amazon web services (aws) glue," *International Journal of Management, IT and Engineering*, vol. 8, pp. 108–122, 2018.

[53] T.-C. Chiu *et al.*, "Semisupervised distributed learning with non-iid data for aiot service platform," *IEEE IoT-J*, pp. 9266–9277, 2020.

[54] "AWS Lambda@Edge," https://aws.amazon.com/lambda/edge/.

[55] Microsoft, "Azure iot edge," https://github.com/Azure/iotedge, 2021.

[56] Baidu, "Openedge," https://github.com/baidu/openedge, 2021.

[57] "Kubernetes Manual," 2017.

[58] D. Huba *et al.*, "Papaya: Practical, private, and scalable federated learning," in *Proceedings of Machine Learning and Systems*, 2022.