INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

Practical Evaluation of Federated Learning in Edge AI for IoT

Sauryadeep Pal [a,*], Muhammad Umair [a], Wooi-Haw Tan [a], Yee-Loo Foo [a]

[a] *Faculty of Engineering, Multimedia University, Cyberjaya, 63100, Malaysia*

Corresponding author: [*]ylfoo@mmu.edu.my

***Abstract*— AI running locally on IoT Edge devices is called Edge AI. Federated Learning (FL) is a Machine Learning (ML) technique that builds upon the concept of distributed computing and preserves data privacy while still supporting trainable AI models. This paper evaluates the FL regarding practical CPU usage and training time. Additionally, the paper presents how biased IoT Edge clients affect the performance of an AI model. Existing literature on the performance of FL indicates that it is sensitive to imbalanced data distributions and does not easily converge in the presence of heterogeneous data. Furthermore, model training uses significant on-device resources, and low-power IoT devices cannot train complex ML models. This paper investigates optimal training parameters to make FL more performant and researches the use of model compression to make FL more accessible to IoT Edge devices. First, a flexible test environment is created that can emulate clients with biased data samples. Each compressed version of the ML model is used for FL. Evaluation is done regarding resources used and the overall ML model performance. Our current study shows an accuracy improvement of 1.16% from modifying training parameters, but a balance is needed to prevent overfitting. Model compression can reduce resource usage by 5.42% but tends to accelerate overfitting and increase model loss by 9.35%.**

***Keywords*— Internet-of-Things (IoT); edge AI; machine learning; federated learning.**

Manuscript received 15 Nov. 2022; revised 29 Jan. 2023; accepted 2 Sep. 2023. Date of publication 30 Nov. 2023.

I. INTRODUCTION

A. Cloud AI and Edge AI

Artificial Intelligence (AI) can augment Internet of Things (IoT) applications by improving the efficiency of IoT operations, improving data management, and adding new features [1], [2]. Due to the resource demands of AI frameworks, cloud computing is a popular method of implementing AI in IoT systems [3]–[5]. This type of AI, often called Cloud AI, is hosted on the cloud and is not constrained by the limited resources of IoT Edge devices [6], [7]. Cloud computing offers scalable deployments and flexibility in resource allocation [8], which can be quite beneficial for any AI application. However, Cloud AI does have several drawbacks related to speed, response time, and data security [9]. Due to local data collection and processing, Edge AI models are not bottlenecked by large data traffic loads or poor network connections as Cloud AI models have the potential [10]–[13].

Furthermore, centralization means that Cloud AI models are unable to guarantee the security and privacy of their data [14]. A breach in data security can potentially affect all data in the system, including sensitive data such as passwords [15]. Conversely, the data on an edge device is restricted to that device and does not leave it. Even in the case of a security breach, the data on other edge devices is not affected, thus providing a greater degree of data privacy and security [16]–[19]. This is greatly beneficial for IoT applications which can have multiple clients with sensitive data, such as health monitoring systems [20]–[22].

B. Challenges of Edge AI for IoT Applications

While Edge AI can be a beneficial addition to many IoT systems, there are some challenges to overcome when it comes to its implementation on IoT Edge devices:

- Microcontrollers, which are commonly used as IoT Edge devices, have very limited storage and memory [23] thus necessitating a small AI model size and model compression.
- IoT Edge devices also typically have weak CPUs which might struggle or be completely incapable of executing the complex mathematical operations of AI models within a reasonable timescale.
- Locally available data may be limited in terms of both quantity and scope. This inhibits strategies such as model compression [24], along with potentially making the model biased.

Finding solutions for these issues is vital for making Edge AI compatible with a wide variety of IoT applications. One such potential solution is a Machine Learning (ML) technique called Federated Learning (FL).

C. Federated Learning (FL)

FL is a ML technique based on the principles of collaboration and central aggregation [25]. Figure 1 shows the overview of a generic FL arrangement. Each client in the FL network trains locally with its local dataset. In the context of IoT applications, these clients would typically be IoT Edge devices. Post-training, each client then uploads its local model to a central server. The server aggregates the models from all clients and sends the aggregated, updated model back to the clients. FL provides a few key advantages:

- Data privacy is preserved as the data from each client does not leave the client. The federated server only sees a client's ML model parameters, not the data it is trained on.
- Through aggregation, FL allows each client to leverage the training conducted by every other client in the system. This reduces the local training and data each client would require to obtain a similarly performant model.
- FL provides a degree of flexibility to cater to different types of clients. Each client contributes only as much training as their computing resources allow yet can still obtain the most updated and performant model through aggregation. This can be quite beneficial for IoT applications with IoT Edge devices with varying computing capabilities.
- Implementing FL on existing Edge AI systems is easy as FL can be purely software-based except for the Federated Server, which can be implemented on IoT Edge devices such as Raspberry Pi.

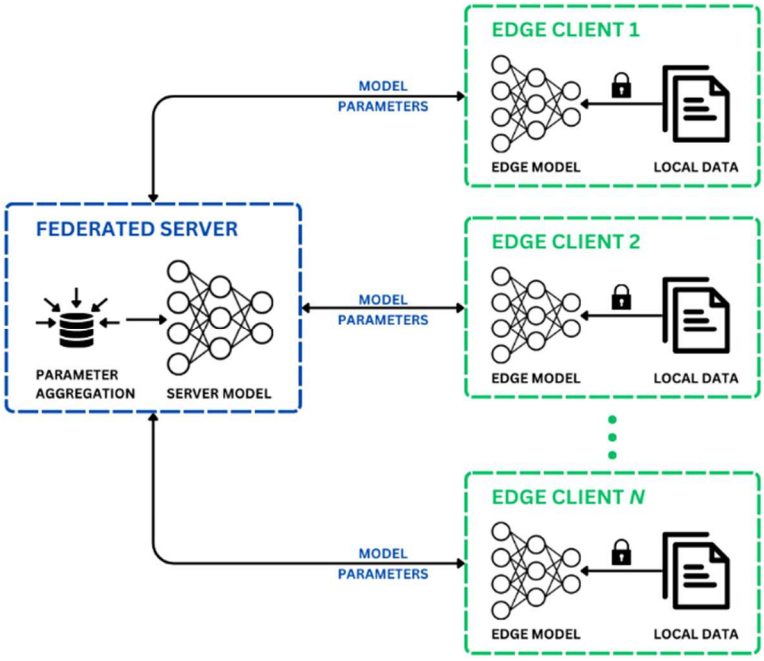

Fig. 1 A generic Federated Learning arrangement

FL is a relatively new technology; currently, much of its research is based on theoretical simulations. While this provides an idea of the strengths and weaknesses of FL, practical evaluations are necessary to judge the true capabilities of FL.

D. Background and Related Works

This section discusses the relevant research on the performance of various Federated Learning algorithms and studies evaluating Federated Learning in practical IoT scenarios. The data collected in an IoT system typically tends to display heterogeneity statistically and within the IoT system as a whole. In an IoT system, the statistical distribution of the data collected by edge devices can greatly vary from device to device. Some devices may collect very few data samples, while others may collect samples biased in a specific direction. This leads to the overall data in the system being Non-IID (Non-Independently and Identically Distributed). Non-IID data has been known to degrade FL performance by reducing model accuracy and making model convergence inconsistent, requiring more training to achieve global model convergence [26]. For example, FedAvg, one of the most basic FL algorithms, can show up to ~55% reduction in accuracy when trained on non-IID data [27].

Furthermore, IoT edge devices may vary greatly in terms of hardware, directly affecting their data collection and ability to perform in FL training sessions. This means that devices that are consistently more active in FL training can make the model more biased towards themselves. This phenomenon is known as system heterogeneity, wherein an imbalance in device capabilities negatively affects the performance of an FL-trained model. Several FL algorithms, such as FedProx, have been developed to improve the performance of FL in the presence of heterogeneity by stabilizing and guaranteeing model convergence [28]. However, comprehensive research on the performance of these algorithms indicates that they are

more sensitive to heterogeneity than initially claimed. Evaluation of these algorithms shows that their performance degrades in a statistical and system heterogeneity setting, and model divergence is uncommon [29]. This is especially problematic for IoT, as most applications would prefer adaptable and flexible solutions when implementing Edge AI in IoT systems.

In a practical IoT scenario, IoT Edge devices will typically have responsibilities other than FL model training and must allocate time and computing resources to fulfill these responsibilities. Therefore, finding a tradeoff between training time and model performance is important by choosing the most optimal training epochs for edge devices and the data batch size when training [29]. Evaluation of FL in a practical IoT setting [30] shows that model training incurs significant resource costs in CPU usage, power usage, and network overhead. Furthermore, even on a relatively powerful IoT edge device such as the Raspberry Pi 3, this study could not feasibly train complex neural network models such as MobileNet V1 on the device.

II. Material and Method

We propose several solutions to solve the drawbacks of FL for IoT applications, as identified by previous works. A simple solution to degrading FL performance in heterogeneous environments is to make the FL server dynamic. For instance, the server could track the average model accuracy and loss over n training rounds and halt training if the model performance degrades by a certain percentage. Another identified issue is the high computing cost of running complex ML models on IoT edge devices which makes inference extremely time consuming and even precludes some models from running on certain IoT Edge devices. Model optimization is therefore necessary and can be achieved through model compression. In this study, we shall implement three specific model compression techniques: pruning, clustering, and quantization, and observe their effect on model performance and resource usage.

A. Federated Learning Setup

The model used for experimentation is a multi-class classification Convolutional Neural Network (CNN) trained on the Dry Bean dataset [31]. The KerasTuner tool is used to obtain an initial optimized model architecture. The FL framework consists of a local Edge server and multiple Edge clients. Figure 2 provides an overview of the FL framework setup.

The Edge server and clients are connected through a wireless Local Area Network (LAN), specifically a WiFi Access Point. The server provides two services through separate ports:

- A training service that coordinates Federated Training between participating Edge clients. This service executes the FedAvg algorithm to aggregate model parameters from clients and broadcasts the aggregated parameters back to the clients. It also keeps track of resource usage during training.
- An updated model provider service that serves the most updated model via HTTP. This service allows Edge client devices incapable of training to still get the most updated model through a simple HTTP request.

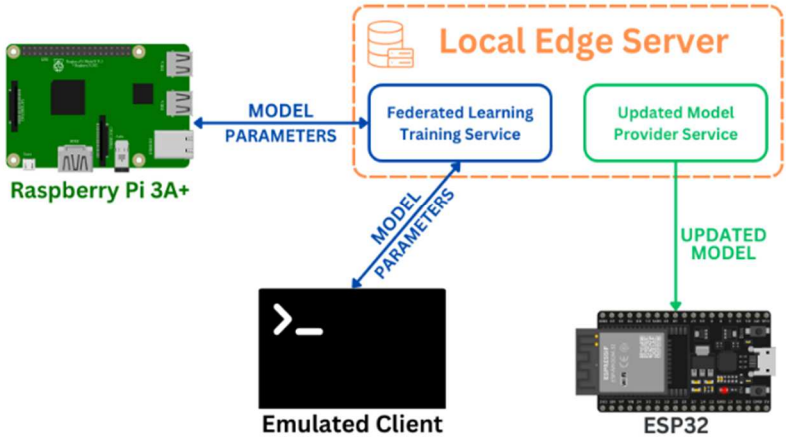

Fig. 2 Federated Learning setup overview

The Edge clients are IoT Edge devices of varying capabilities. There are three types of IoT Edge devices taking part in the experiments:

- **Software emulators are essentially used to emulate additional clients for FL and are capable of** inference and training. They interact with the server through the training service. In an actual IoT scenario, they represent the most computationally capable IoT Edge devices, such as smartphones.
- **Raspberry Pi 3 Model A+:** This Raspberry Pi is a moderately powerful IoT Edge device and can both infer and train with its local model. Similar to the emulated clients, it interacts with the server through the training service. As a commonly used IoT Edge device, the Raspberry Pi provides a good benchmark of FL performance in a typical IoT application.
- **ESP32:** The ESP32 is a low-power microcontroller and is very common in IoT applications. It represents those IoT Edge devices that have very limited computing resources and are able to only infer, and not train, with their local model. The ESP32 cannot participate in training, so it instead gets the most updated model through the server's model provider service.

B. Experimental Setup

We used Python and multiple Python libraries for this project to set up the test environment. The Python version used is 3.10.7, 64-bit Windows. All ML models are built with TensorFlow v2.12.0 and Keras, and the Flower library v1.3.0 implements FL.

The two IoT Edge devices used for testing are the Raspberry Pi 3 Model A+ and the ESP32. The Raspberry Pi is running Python 3.9.16, whereas the ESP32 is programmed with the ESP-IDF. CUDA is not available for any of the IoT Edge devices.

A laptop is used as the server for FL and for instantiating software-emulated IoT Edge devices. This laptop has an Intel i5-8300H CPU, GTX 1060 GPU, and 8 GB of RAM. The operating system is Windows 11 64-bit.

C. Federated Training Round Overview

Figure 3 shows the overview of one round of Federated Training. Each round begins with the server asking all participating clients to start local training and sending optional training instructions to each client, such as the epochs to train for. Once every client has completed training and sent the updated parameters to the server, it will aggregate these parameters using the FedAvg algorithm. The aggregated parameters are applied to the server's own model and also broadcast back to the clients so that they may update their local models. Alongside the weights, each client also sends the resource usage recorded during training to the server. This includes the CPU time utilized and the RAM usage during training, and these metrics are averaged and recorded by the server. After the server and clients have updated their models, they evaluate their models with their local datasets.

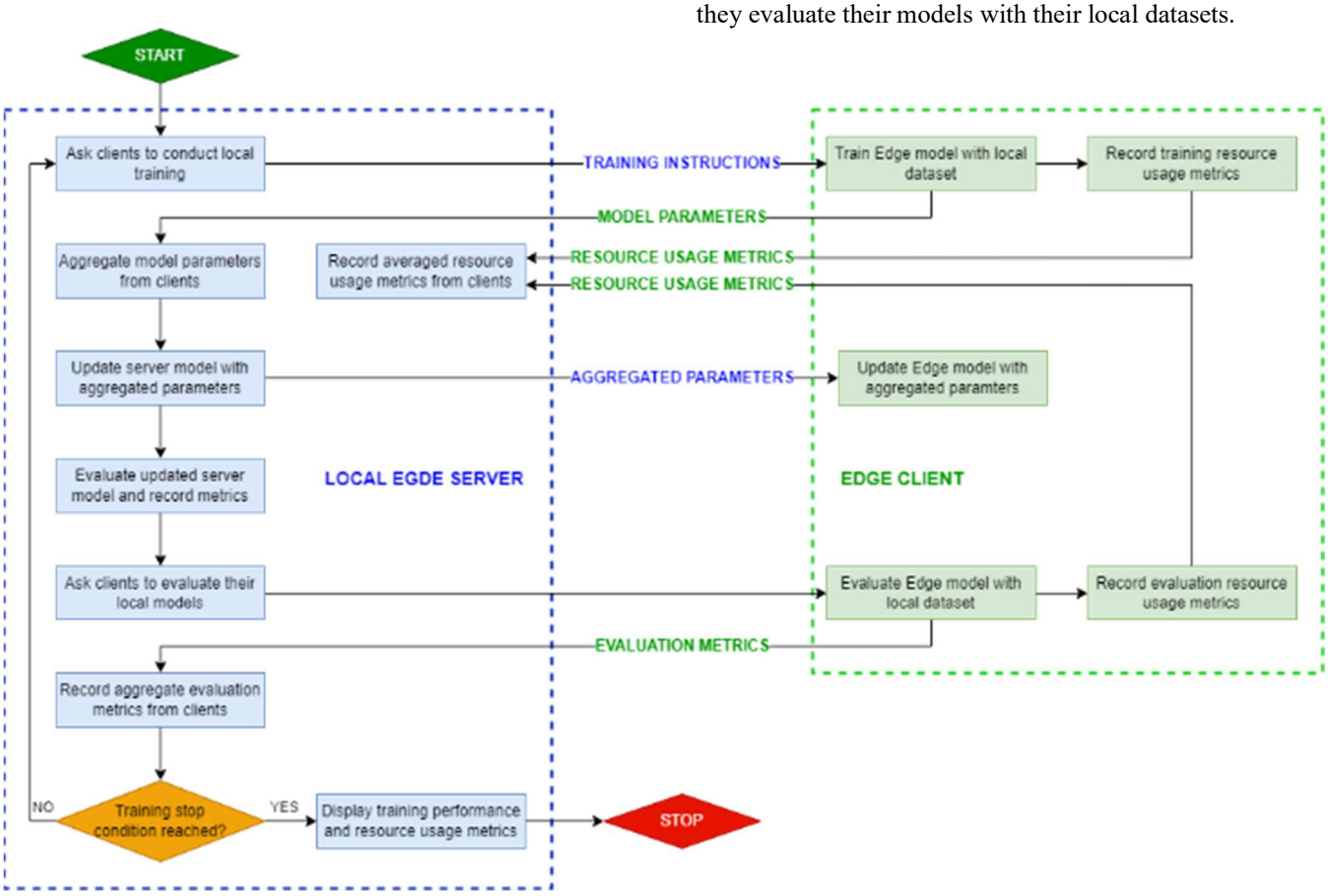

Fig. 3 Process flow of one round of Federated Training

The clients send their evaluation results to the server in terms of loss and accuracy, along with the resource usage metrics for evaluation. The server records a weighted average of the accuracies and losses (the weight being the number of samples in a client's dataset) and adds the averaged evaluation resource usage to the averaged training resource usage. Lastly, the server records the time elapsed to complete the training round. This concludes one round of training.

Two conditions determine if the server should continue with the next round of training:

- If the pre-determined number of training rounds has already been completed, or
- If further training will degrade model performance.

If either of these criteria are met, the server stops training after the current round has concluded. It then displays the relevant training results, as can be seen in Figure 4, and stores every recorded metric in a CSV file. This file can later be used to evaluate the overall effect of the training session in greater detail.

```
FEDERATED TRAINING RESULTS
--------------------------------------------------------------------------------

Training info:
        Num Rounds = 10         Num Clients = 5

Before training:
        Client loss = 2.16203243        Client accuracy = 0.11192661
        Server loss = 2.18900585        Server accuracy = 0.10005877

Training (10 rounds):
Round   Server Loss     Server Accuracy         Client Loss     Client Accuracy
--------------------------------------------------------------------------------
001     0.68662101      0.74346167              0.69903720      0.74128440
002     0.46223995      0.88289744              0.48413841      0.87559633
003     0.36453977      0.90611225              0.37992977      0.89614679
004     0.31188771      0.91610342              0.32696136      0.91009174
005     0.27742761      0.92109901              0.29762688      0.91192662
006     0.25628492      0.92359680              0.27986749      0.91412843
007     0.25031903      0.92300910              0.26853178      0.91522935
008     0.23284981      0.92565382              0.26014436      0.91743118
009     0.23551476      0.92403764              0.25441688      0.91669725
010     0.22860996      0.92991477              0.25036937      0.91559633
--------------------------------------------------------------------------------
```

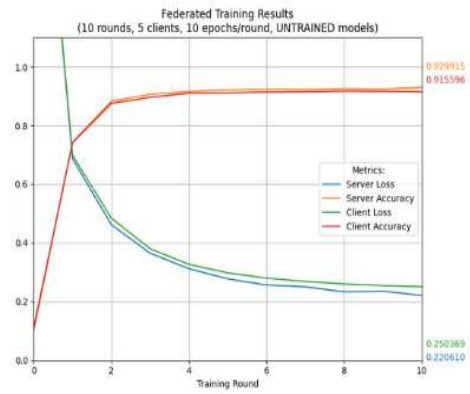

Fig. 4 The console and graphical output of a training session, showing loss and accuracy metrics

D. *Measuring FL Performance and Resource Usage During Training*

To evaluate the performance and resource usage of FL, we measure the following metrics during each training round:

- **Client Loss and Accuracy**: After training and model update during each training round, all clients execute a single round of evaluation on a fraction of their local dataset. Their evaluated losses and accuracies are aggregated via a weighted average (where the weight is the number of samples a client has in their local dataset) and reported as the client's loss and accuracy. These two metrics are also collectively referred to as the client metrics.
- **Server Loss and Accuracy**: The FL server has a dataset concatenated from all participating clients' datasets. In an actual FL setup, the server will most likely not have this dataset or, ideally any client data, as data privacy is a core tenet of FL. We have provided the server with this dataset only to facilitate FL evaluation. The server also instantiates a copy of the ML model running on all Edge clients but only uses it for evaluation and not training. Each training round, the server randomly samples a portion of the dataset and runs a single round of evaluation with its own copy of the model, after updating said model parameters with the aggregated model parameters. The evaluated loss and accuracy are recorded.
- **Average Training Resource Usage**: The psutil (process and system utilities) Python library is used to measure the accumulated CPU time for Federated Training in each participating client. This library is also used to measure the estimated RAM usage during training. These two measurements are averaged across clients to get a measure of the CPU time and RAM usage during each round of FL training. The server also records the time elapsed for each round of training, using the timeit Python library.

E. *Federated Training Evaluation*

To counter the effects of overfitting and other types of model performance degradation, we propose implementing a training evaluation system to track the effect of training while a training session is on-going. Every *n*-th round of training, the server calculates the average of the server loss and accuracy over the current and past *n*-1 rounds. These metrics are used to determine whether FL is deteriorating or improving the model performance and to take action accordingly. Currently, the action is to halt training.

Figure 5 shows a side-by-side comparison of this system in action, with the left graph being the control. Both of these training sessions have the same number of clients, with each being provided a biased dataset. These datasets contain samples from only four randomly chosen classes of the seven in the original dataset. Additionally, both training sessions have the following starting conditions:

- 50 planned training rounds
- 10 clients with identical computing resources
- 2 local epochs of training per client during each training round
- Using the same pre-trained model for both server and clients

For the graph on the right, the training evaluation rounds is set to multiples of 5. Hence every 5n-th round (where $n > 0$), the average server loss and accuracy over the current and past 4 rounds is calculated. If the average loss and accuracy have both deteriorated from the previous average, the system assumes that training is making the model worse and stops further training. Hence, the training stops after only 10 rounds out of the planned 50 rounds.

F. *Manipulating Training Parameters to Improve FL Performance*

In a practical IoT scenario, we can readily manipulate three Federated Training parameters: the number of training rounds, the number of local training epochs, and the number of clients participating in training. Changing model hyperparameters, such as the learning rate, would have a bigger impact on FL performance. However, most Edge AI models are usually static and not easily modifiable. Thus, in this project, we instead experiment with changing the number of training rounds, the number of local training epochs, and the number of clients in a training session. For these experiments, we use emulated software clients to ensure that each client is identical and that diversity does not affect the results. To prevent potential bias from differing model parameters and internal model structure, each client and the server are provided with the same, completely untrained model with no type of model compression applied. The server has the entire Dry Bean dataset for evaluation, whereas each client is provided with a random sub-sample of the Dry Bean dataset. Each client dataset has approximately the same

number of samples, and each client uses their local dataset for training and evaluation. When one training parameter is being experimented on, the others are kept constant. The training evaluation system described in the previous section has also been disabled for these experiments. The metrics measured for these experiments are the server loss and accuracy and the client loss and accuracy. To get an idea of the model performance at the end of training, we measure the average of these metrics over the last 5 rounds of training.

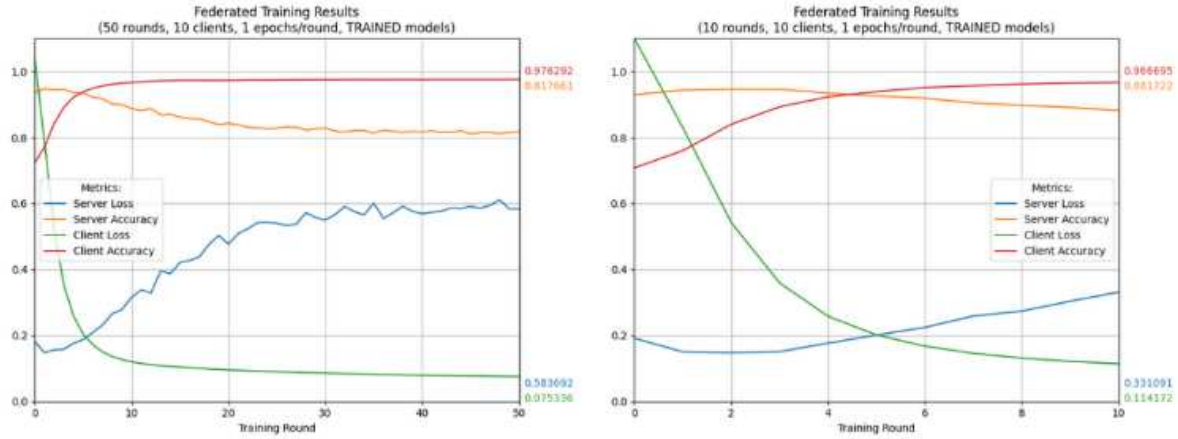

Fig. 5 The outcome of Federated Training without (left) and with (right) training evaluation. Note that these graphs also show the pre-training (round 0) metrics

Figure 6 shows the results of adjusting the training rounds. We started with 50 training rounds and went up in steps of 50 up to 300 training rounds. There were 10 participating clients for all training sessions, with each client training locally for 1 epoch per training round.

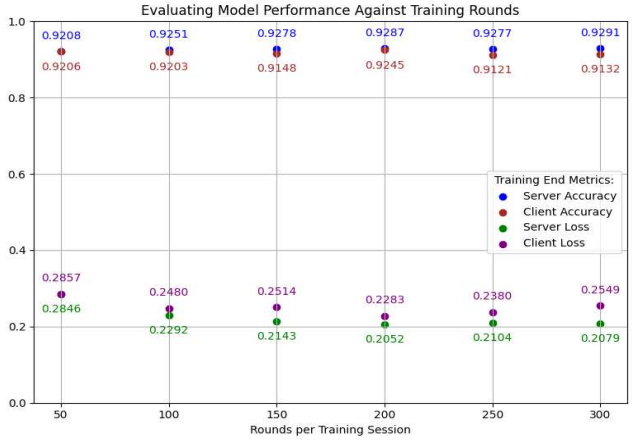

Fig. 6 End of training loss and accuracy metrics when varying training rounds

Figure 7 shows the results of adjusting the local training epochs. We started with 1 epoch per round and went up to 10 epochs per round. The number of participating clients was 10 for all training sessions. The number of training rounds was also kept at a constant 50 for all training sessions. Figure 8 shows the results of varying the number of clients participating in training. We started with 5 clients and went up to 15. The local training epochs per client was fixed at 1, and the number of training rounds was also fixed at 50 for all training sessions.

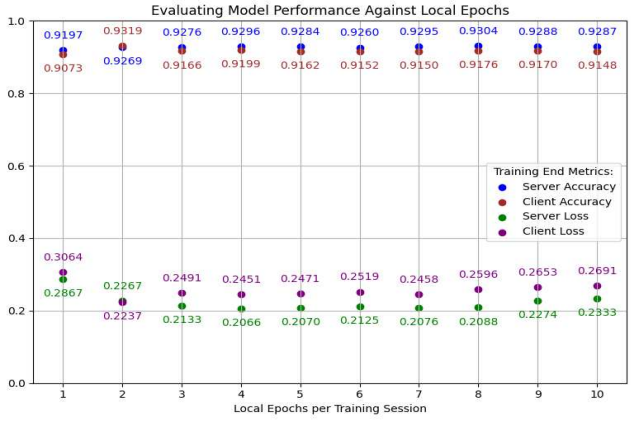

Fig. 7 End of training loss and accuracy metrics when varying local training epochs

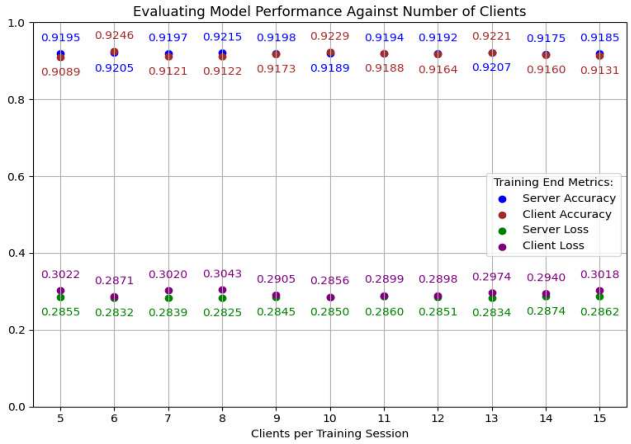

Fig. 8 End of training loss and accuracy metrics when varying the number of participating clients

G. Implementing Model Compression

Model compression is the generic name for a variety of techniques that can help reduce the size and computational complexity of a neural network. These reductions can be the difference that allows an IoT Edge device to run a particular model, and hence are an important consideration for any Edge AI system. With regards to FL, a less complex model can speed up training as local training would require fewer computing resources.

In this project, we have implemented three model compression techniques. It should be noted that these model compression techniques apply only to the model and not any of its training and evaluation code and data. The techniques are as follows:

- **Pruning**: Neural network pruning is a technique by which parameters are systematically removed from a large, over-parameterized model, to produce an overall smaller, less complex model [32]. Any drop in accuracy caused by pruning is usually rectified by tuning the pruned model with the original's model dataset. This project uses the TensorFlow Model Optimization Toolkit to implement pruning. The pruning algorithm used is constant sparsity, so a constant user-defined sparsity is targeted for the parameters of each layer. The target sparsity for this project is 75%, so the pruning algorithm will attempt to remove 75% of the parameters in each layer. It should be noted that this type of pruning does not change the model structure by physically removing model parameters. Rather, it "masks" parameters by setting their value to zero. This way, any multiplication operations with these parameters will have a pre-determined answer of zero and the CPU does not need to perform the actual multiplication, thus reducing the CPU usage. Furthermore, compression algorithms such as gzip can take advantage of the large number of continuous zeroes to compress the model more efficiently.
- **Clustering**: Clustering is based on the k-Means clustering algorithm and is contingent on the idea of weight sharing. The goal is to group similar weights in each neural network layer into clusters and represent each weight in a cluster by its centroid value [33]. Clustering allows compression algorithms to take advantage of the data redundancy introduced by several unique weights now represented by the same value to reduce model size greatly. This project uses the TensorFlow Model Optimization Toolkit to implement clustering. The target number of clusters per layer is set to 3. As with pruning, clustering does not change the underlying model structure but rather manipulates existing parameters.
- **Quantization**: Quantization is the process of reducing the number of bits used to represent model parameters. For example, model weights can be stored as 16-bit floating points instead of 32-bit, halving their storage requirement. Aside from storage savings, quantization schemes can also reduce the computational complexity of the model. Microcontrollers, particularly other low-power IoT Edge devices, benefit greatly as they typically have low-power CPUs and limited memory. Thus, any reduction in complexity will result in a speed-up in inference and training. We implement float16 quantization in this project using the TensorFlow Lite model conversion toolkit.
  This quantization scheme attempts to convert as many model parameters as possible from 32-bit floating points to 16-bit floating points, which greatly reduces the storage occupied by the model.

This experiment uses a single Raspberry Pi 3 Model A+ as the only client for 20 rounds of Federated Training. Both the server and the client use completely untrained models and train locally for 5 epochs each training round. The server has the entire Dry Bean dataset to evaluate its model, while the client is provided a randomly sub-sampled portion of the Dry Bean dataset for training and evaluation. Table 1 shows the effect of various model compression techniques on model performance during Federated Training.

TABLE I
MODEL PERFORMANCE METRICS WHEN IMPLEMENTING VARIOUS TYPES OF MODEL COMPRESSION

Compression Type	Server Loss	Client Loss	Server Accuracy	Client Accuracy	Loss Deviation (%)	Accuracy Deviation (%)
No Compression	0.240334484	0.238781825	0.922480166	0.928205132	0.648	0.619
Pruning, Clustering	0.256751227	0.243673545	0.923626208	0.931868136	5.227	0.888
Quantization	0.246050501	0.200120828	0.919806063	0.932600737	20.588	1.381
Pruning, Quantization	0.236408308	0.172866923	0.921716142	0.951648355	31.051	3.196
Clustering, Quantization	0.243828276	0.189846137	0.921187186	0.941391945	24.895	2.17
Pruning, Clustering, Quantization	0.262808117	0.222650793	0.92356745	0.932600737	16.544	0.973

The metrics here are averaged over the last 5 rounds of a training session. The accuracy and loss deviation columns show the difference between server and client metrics. As the server has the entire dataset for evaluation, these deviations can indicate whether the model's performance is deteriorating. Table 2 shows the uncompressed and compressed sizes of the model file when using the gzip file compression algorithm. Model size is important, as storage can be a premium in many IoT Edge devices.

TABLE II
UNCOMPRESSED AND GZIP COMPRESSED MODEL FILE SIZES

Compression Type	Uncompressed File Size (Bytes)	Compressed File Size (Bytes)	Compression Ratio
No Compression	631220	278525	2.266
Pruning, Clustering	629492	48741	12.915
Quantization	355740	148755	2.391

Pruning, Quantization	356116	148871	2.392
Clustering, Quantization	354228	44092	8.034
Pruning, Clustering, Quantization	354228	44082	8.036

Figure 9 shows the average resource usage per training round in terms of CPU time, RAM usage, and the time required for one round of training. It should be noted that these resource measurements are estimations and may not correspond exactly to the actual resource usage during training. While all effort was made to conduct the training in an isolated environment with no other programs running in the foreground, the Raspberry Pi's OS may have been running background tasks that could have affected the resource usage measurements. Regardless, these estimates still provide a good approximation of the resource usage during training.

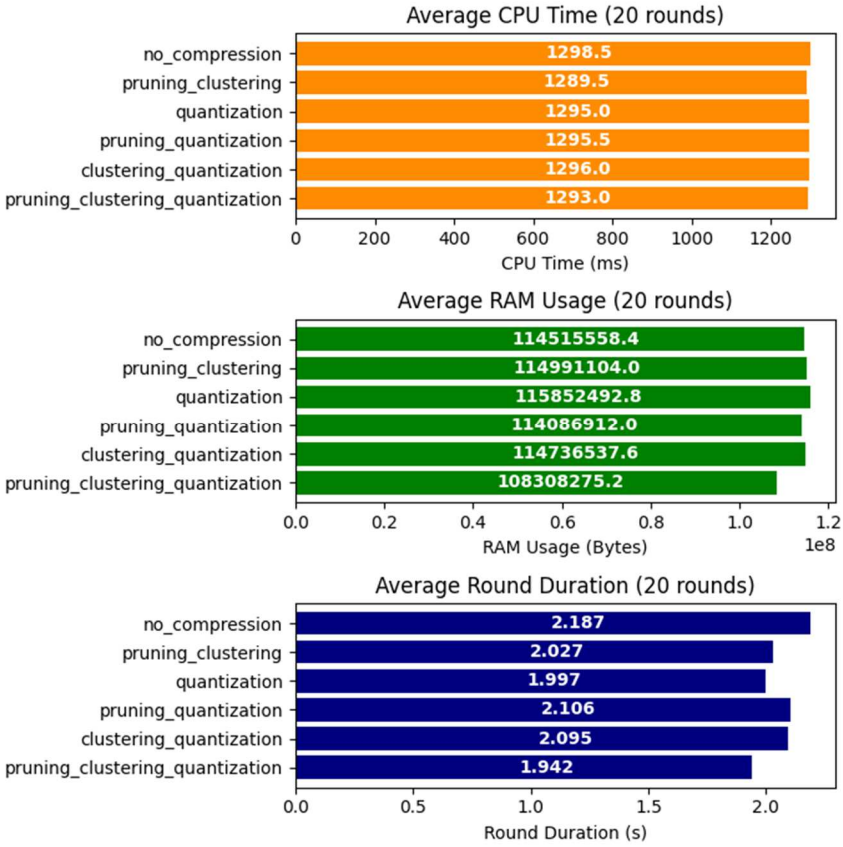

Fig. 9 Training resource usage when implementing various types of model compression

III. Result and Discussion

A. Minimizing Model Degradation due to heterogeneity

Previous works mostly focused on using more sophisticated FL algorithms to counter degrading model performance with heterogeneous data. However, research has shown that these algorithms do not perform to the level expected, so our approach is to make the FL system dynamic and able to respond appropriately when it detects worsening model performance. Our proposed implementation consists of a detection system based on averaging model performance metrics over a fixed number of rounds and a default response of simply stopping further training.

From Figure 5, evaluating both training sessions' final loss and accuracy metrics shows that the early training stop has successfully reduced model degradation. The client metrics of both sessions might give the incorrect impression that the model is performing well and is soon to converge. As each client has a biased dataset, training and evaluating a model with the same dataset will naturally produce good loss and accuracy metrics with the bias going unnoticed. However, the server has an unbiased dataset, so its model evaluation shows the true effect of training. The server loss and accuracy in the control training session reduces heavily with more training. The session with training evaluation stops training early, thus preventing the model from being degraded as much. The final evaluated server loss is 43.28% less than the control session, while the server accuracy is 7.835% more.

Our current system, while serving its designed purpose, is quite basic at this time. However, the system is scalable and can be made more sophisticated with research. For example, future projects may focus on more complex responses, such as adjusting the local training epochs for the clients or reducing the number of clients participating. A major drawback of a dynamic training system such as this is that while it can be quite effective, it depends on the server's accuracy and loss metrics for unbiased training evaluations. Thus, the server would need to have a large and varied dataset, with the best-case scenario being to have a dataset concatenated from the datasets of all clients in the Edge AI framework. Any client data leaving its host client can be a major data security and privacy breach, especially for sensitive applications such as smart healthcare. The risk can

be somewhat mitigated by anonymizing client data so that its origin is unidentifiable, or by using publicly available datasets. Both methods have their drawbacks, but the point remains that a reduction in data security and privacy is something to keep in mind when implementing a dynamic training system.

B. Evaluating the Effect of Varying Training Parameters in Federated Learning

We experimented with varying three training parameters, the number of training rounds, the local training epochs, and the number of clients participating in training, and observed the effect that each of these had on Federated Training. Figures 6, 7, and 8 show the results of these experiments. Before coming to any conclusions, it is important to note that these results must be assumed specific to our FL environment without further study. It is possible that the results here may not be reproducible in other FL environments that differ in terms of the type of ML model used or a number of other factors. That being said, we believe that the results of these experiments expose some patterns that could be common in most, if not all, FL applications. The following observations are made:

- Regarding the number of training rounds, we can observe from Figure 6 that the model does become more accurate and less lossy with more training. However, after a certain point, this is no longer the case. The best-recorded accuracy and loss metrics are when the model is trained for 200 rounds. Beyond that, the accuracy starts decreasing, and the loss increases, which indicates the model is not converging. Furthermore, the deviation between server and client metrics becomes noticeably larger. When the model is trained for 350 rounds instead of 200, the difference between server and client loss goes from 0.0231 to 0.047, a 103.46% increase. These are signs that the model could be overfitting. Thus, the conclusion here is that training a model indefinitely will be detrimental to its performance. Instead, an optimal number of training rounds should be found, also keeping in mind other factors that have not been explored here, such as the training time and resource usage.
- There is a similar pattern with local training epochs. While 1 epoch seems insufficient, the optimal number at least for our use case, seems to be 2. Training for more epochs is more resource-intensive and time-consuming, making the model perform worse and inhibiting model convergence. This can be observed in Figure 7, wherein the accuracy and loss degrade with local epochs > 2, along with an increased deviation in both metrics. For the training session with 10 local epochs, the deviation in accuracy is 178% more and the deviation in loss is 1093.33% more as compared to the session with 2 local epochs. This is once again a potential sign of overfitting. As with the number of training rounds, careful considerations must also be made when selecting the optimal number of local training epochs.
- The number of clients seems to have the least effect on Federated Training. This experiment could be due to the variance in the number of clients and the maximum number of clients being quite low. A practical IoT scenario could have thousands of clients; in that context, only 15 may not be enough to affect training. Regardless, hints of the same pattern are still observed with training rounds and local training epochs. When comparing the training session with 10 clients and the session with 15 clients, the loss deviation increases by 0.0096. A similar increase of 0.0161 is also observed when the session with 5 clients. Thus, too few clients results in insufficient training, and too many clients will most likely make the training detrimental to the model in terms of both performance as well as resource usage.

The overarching theme of these results is that finding a balance is necessary during Federated Training. The maximum observed increase in accuracy is 1.16% when the local training epochs are increased to 8 from 1. However, this same change also increases the loss deviation by almost 158%. Too much of any parameter, be it the number of training rounds, the local training epochs, or the number of clients, has the tendency to make the model worse by making it less likely to converge and increasing the chance of overfitting, thus defeating the entire purpose of training. Simultaneously, too little training may not be sufficient to improve the model in an observable manner. However, finding this balance could be an issue as the balancing point will differ greatly from use case to use case. At this time, the best solution seems to be doing something similar to the experiments discussed here: to run trials with different values and then select the most optimal ones.

C. Evaluating Model Compression Techniques for More Accessible Federated Learning

We implemented and evaluated the effectiveness of three model compression techniques in Federated Training: pruning, clustering, and quantization. As far as we are aware, this project is the first of its kind to implement and evaluate the effects of model compression in an FL environment. From the results in Table 1, Table 2 and Figure 9, the following is observed:

- The model with no compression has the most robust performance. It has the least deviation in loss and accuracy out of all the models and is thus the most resistant to overfitting with continued training. However, it also has the largest model size and file compression does not reduce the original model file size by much.
- Every model compression technique, implemented individually or in combination with others, causes an observable degradation in model performance, and a minor increase or decrease in the usage of one or more computing resources. The biggest difference is seen in training time, in which up to 11.2% reductions can be observed.
- Pruning and clustering do not change the model size but do greatly improve the efficacy of file compression algorithms. Compared to the uncompressed model, the compression ratio increased from 2.266 to 12.915 (a 469.947% increase). A smaller-sized compressed model requires less storage and is easier to transmit over networks, which is advantageous for both Edge AI and FL. However, pruning and clustering does put the

model at greater risk of overfitting earlier. The accuracy deviation for the pruned and clustered model, for example, is approximately 43.5% more as compared to the uncompressed model.

- Quantization seems to have the biggest effect in reducing model size. This decrease in model size is especially important in IoT as some Edge devices, such as the ESP32 do not have the required memory to load large models. The drawback of quantization is that the loss in precision when converting parameters from single precision to half-precision floating points causes a drop in robustness, with the quantized model having noticeably higher loss and accuracy deviations compared to the uncompressed model. The deviations get even worse when quantization is used in conjunction with any other compression method, with the exception being when quantization, pruning, and clustering are implemented together. Lastly, the CPU time and RAM usage decrease is minimal or even non-existent in any compression involving quantization. One possible cause for this could be the extra overhead from the device having to truncate full-precision training parameters to half-precision model parameters.
- The pruned, clustered, and quantized model appears to have the best resource savings. Compared to the uncompressed model, the CPU time is decreased by approximately 0.42%, RAM usage is reduced by 5.42%, and each round takes approximately 11.2% less time to complete. Accuracy-wise, this model outperforms the uncompressed model, although it is more lossy. This compression scheme makes the model more prone to overfitting, and the server loss is also 9.35% more than the uncompressed model.

Overall, model compression is an effective technique for reducing model size and complexity. For IoT applications, model compression, particularly quantization, may be necessary to make some models accessible to low power IoT Edge devices. We tested this out by stripping the model of its training and evaluation portions and keeping just the base model. Furthermore, we changed the quantization scheme to full integer, wherein all model parameters are converted to 8-bit or 16-bit integers [34]. In this case, the uncompressed model was 282,409 bytes and the quantized model was 74,977 bytes in size. Our ESP32 fails to load the un-quantized model as it does not have enough free memory to allocate but is able to load and successfully infer with the quantized model. For devices already capable of inference, model compression can reduce resource usage during training to various degrees thus allowing the device more computational resources for other tasks.

The primary drawback of model compression is a drop in model performance. The observed increases in loss and accuracy deviations for this particular experiment were not insignificant, and this is still a comparatively simple deep learning model compared to those used in image recognition or other complex tasks. There will always be a tradeoff between model performance and resource usage when implementing model compression. The drop in model performance will vary across use cases and concerning FL, will also depend on factors such as local training epochs and the number of training rounds. It is up to the user to decide whether the resource usage decrease and storage savings brought about by implementing one (or more) model compression techniques is worth the reduction in model performance.

IV. Conclusion

In this work, we evaluated FL regarding model performance and resource usage in a practical scenario, and implemented a few solutions to improve FL. One key weakness of FL is degrading model performance in the presence of heterogeneity. Our solution is not to modify the underlying FL algorithm but to make the training framework dynamic enough to detect and respond to deteriorating model performance. This technique worked, although at the cost of reduced data privacy and security. Furthermore, we evaluated the effect of training parameters on FL performance. The results reveal that careful balancing of training parameters, such as the training rounds, local training epochs, and the number of participating clients, is required. Poorly chosen parameters can lead to insufficient training or detrimental effects on the model's loss and accuracy. Lastly, we experimented with implementing various types of model compression in order to make FL, and Edge AI in general, more accessible to low-power IoT Edge devices. A reduction in computing resource usage was certainly observed with model compression, and in one case, it enabled a low-power IoT Edge device to infer with a model it was previously incapable of running. Model compression did, however come with its drawbacks, namely worse model performance.

References

[1] P. K M and N. B S, "AIoT - Artificial Intelligence of Things," *Int. J. Adv. Res. Sci. Commun. Technol.*, pp. 88–92, Apr. 2021, doi: 10.48175/ijarsct-v4-i3-013.

[2] A. Matin, M. R. Islam, X. Wang, H. Huo, and G. Xu, "AIoT for sustainable manufacturing: Overview, challenges, and opportunities," *Internet of Things*, vol. 24, p. 100901, Dec. 2023, doi: 10.1016/J.IOT.2023.100901.

[3] Y. Wu, "Cloud-Edge Orchestration for the Internet of Things: Architecture and AI-Powered Data Processing," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12792–12805, Aug. 2021, doi: 10.1109/JIOT.2020.3014845.

[4] F. Firouzi, B. Farahani, and A. Marinšek, "The convergence and interplay of edge, fog, and cloud in the AI-driven Internet of Things (IoT)," *Inf. Syst.*, vol. 107, p. 101840, Jul. 2022, doi: 10.1016/J.IS.2021.101840.

[5] U. F. Mustapha, A. W. Alhassan, D. N. Jiang, and G. L. Li, "Sustainable aquaculture development: a review on the roles of cloud computing, internet of things and artificial intelligence (CIA)," *Rev. Aquac.*, vol. 13, no. 4, pp. 2076–2091, Sep. 2021, doi: 10.1111/RAQ.12559.

[6] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A Survey on End-Edge-Cloud Orchestrated Network Computing Paradigms," *ACM Comput. Surv.*, vol. 52, no. 6, Oct. 2019, doi: 10.1145/3362031.

[7] C. Mwase, Y. Jin, T. Westerlund, H. Tenhunen, and Z. Zou, "Communication-efficient distributed AI strategies for the IoT edge," *Futur. Gener. Comput. Syst.*, vol. 131, pp. 292–308, Jun. 2022, doi: 10.1016/J.FUTURE.2022.01.013.

[8] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Gaithersburg, MD, 2011. doi: 10.6028/NIST.SP.800-145.

[9] D. Saadia, "Integration of Cloud Computing, Big Data, Artificial Intelligence, and Internet of Things: Review and Open Research Issues," *https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/IJWLTT.2021010102*, vol. 16, no. 1, pp. 10–17, Jan. 2021, doi: 10.4018/IJWLTT.2021010102.

[10] G. K. Sriram, “Edge Computing Vs. Cloud Computing: an Overview of Big Data Challenges and Opportunities for Large Enterprises,” *www.irjmets.com @International Res. J. Mod. Eng.*, vol. 04, no. 01, pp. 1–6, 2022.

[11] L. Sun, L. Sun, X. Jiang, H. Ren, H. Ren, and Y. Guo, “Edge-Cloud Computing and Artificial Intelligence in Internet of Medical Things: Architecture, Technology and Application,” *IEEE Access*, vol. 8, pp. 101079–101092, 2020, doi: 10.1109/ACCESS.2020.2997831.

[12] S. Duan *et al.*, “Distributed Artificial Intelligence Empowered by End-Edge-Cloud Computing: A Survey,” *IEEE Commun. Surv. Tutorials*, vol. 25, no. 1, pp. 591–624, 2023, doi: 10.1109/COMST.2022.3218527.

[13] F. Saeik *et al.*, “Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions,” *Comput. Networks*, vol. 195, p. 108177, Aug. 2021, doi: 10.1016/J.COMNET.2021.108177.

[14] A. Singh, S. C. Satapathy, A. Roy, and A. Gutub, “AI-Based Mobile Edge Computing for IoT: Applications, Challenges, and Future Scope,” *Arab. J. Sci. Eng. 2021 478*, vol. 47, no. 8, pp. 9801–9831, Jan. 2022, doi: 10.1007/S13369-021-06348-2.

[15] K. Cao, Y. Liu, G. Meng, and Q. Sun, “An Overview on Edge Computing Research,” *IEEE Access*, vol. 8, pp. 85714–85728, 2020, doi: 10.1109/ACCESS.2020.2991734.

[16] Z. Xu, W. Liu, J. Huang, C. Yang, J. Lu, and H. Tan, “Artificial Intelligence for Securing IoT Services in Edge Computing: A Survey,” *Secur. Commun. Networks*, vol. 2020, 2020, doi: 10.1155/2020/8872586.

[17] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, “Edge Computing Security: State-of-The-Art and Challenges,” *Proc. IEEE*, vol. 107, no. 8, pp. 1608–1631, Aug. 2019, doi: 10.1109/JPROC.2019.2918437.

[18] H. Zeyu, X. Geming, W. Zhaohang, and Y. Sen, “Survey on Edge Computing Security,” *Proc. - 2020 Int. Conf. Big Data, Artif. Intell. Internet Things Eng. ICBAIE 2020*, pp. 96–105, Jun. 2020, doi: 10.1109/ICBAIE49996.2020.00027.

[19] M. Caprolu, R. Di Pietro, F. Lombardi, and S. Raponi, “Edge Computing Perspectives: Architectures, Technologies, and Open Security Issues,” *Proc. - 2019 IEEE Int. Conf. Edge Comput. EDGE 2019 - Part 2019 IEEE World Congr. Serv.*, pp. 116–123, Jul. 2019, doi: 10.1109/EDGE.2019.00035.

[20] S. S. Ambarkar and N. Shekokar, “Toward Smart and Secure IoT Based Healthcare System,” *Stud. Syst. Decis. Control*, vol. 266, pp. 283–303, 2020, doi: 10.1007/978-3-030-39047-1_13/COVER.

[21] H. A. El Zouka and M. M. Hosni, “Secure IoT communications for smart healthcare monitoring system,” *Internet of Things*, vol. 13, p. 100036, Mar. 2021, doi: 10.1016/J.IOT.2019.01.003.

[22] J. B. Awotunde, R. G. Jimoh, S. O. Folorunso, E. A. Adeniyi, K. M. Abiodun, and O. O. Banjo, “Privacy and Security Concerns in IoT-Based Healthcare Systems,” *Internet of Things*, pp. 105–134, 2021, doi: 10.1007/978-3-030-75220-0_6/COVER.

[23] S. Soro, “TinyML for Ubiquitous Edge AI,” no. 20, Feb. 2021.

[24] T. Sipola, J. Alatalo, T. Kokkonen, and M. Rantonen, “Artificial Intelligence in the IoT Era: A Review of Edge AI Hardware and Software,” *Conf. Open Innov. Assoc. Fruct*, vol. 2022-April, no. i, pp. 320–331, 2022, doi: 10.23919/FRUCT54823.2022.9770931.

[25] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, “A survey on federated learning,” *Knowledge-Based Syst.*, vol. 216, p. 106775, 2021, doi: 10.1016/j.knosys.2021.106775.

[26] T. Liu, J. Ding, T. Wang, M. Pan, and M. Chen, “Towards Fast and Accurate Federated Learning with Non-IID Data for Cloud-Based IoT Applications,” *J. Circuits, Syst. Comput.*, vol. 31, no. 13, pp. 1–10, 2022, doi: 10.1142/S0218126622502358.

[27] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated Learning with Non-IID Data,” *Comput. J.*, no. May, Jun. 2018, doi: 10.48550/arXiv.1806.00582.

[28] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated Optimization in Heterogeneous Networks,” 2018.

[29] M. Tahir and M. I. Ali, “On the Performance of Federated Learning Algorithms for IoT,” *IoT*, vol. 3, no. 2, pp. 273–284, 2022, doi: 10.3390/iot3020016.

[30] Y. Gao *et al.*, “End-to-End Evaluation of Federated Learning and Split Learning for Internet of Things,” *Proc. IEEE Symp. Reliab. Distrib. Syst.*, vol. 2020-Septe, pp. 91–100, 2020, doi: 10.1109/SRDS51746.2020.00017.

[31] M. Koklu and I. A. Ozkan, “Multi-class classification of dry beans using computer vision and machine learning techniques,” *Comput. Electron. Agric.*, vol. 174, p. 105507, Jul. 2020, doi: 10.1016/j.compag.2020.105507.

[32] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” *7th Int. Conf. Learn. Represent. ICLR 2019*, pp. 1–21, 2019.

[33] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, pp. 1–14, 2016.

[34] B. Jacob *et al.*, “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2704–2713, 2018, doi: 10.1109/CVPR.2018.00286.