Ed-Fed: A generic federated learning framework with resource-aware client selection for edge devices

Zitha Sasindran, Harsha Yelchuri, T. V. Prabhakar
Department of Electronic Systems Engineering
Indian Institute of Science
Bengaluru, India, 560012.
Email: {zithas, harshay, tvprabs}@iisc.ac.in

2023 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-8867-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/IJCNN54540.2023.10191316

***Abstract*—Federated learning (FL) has evolved as a prominent method for edge devices to cooperatively create a unified prediction model while securing their sensitive training data local to the device. Despite the existence of numerous research frameworks for simulating FL algorithms, they do not facilitate comprehensive deployment for automatic speech recognition tasks on heterogeneous edge devices. This is where Ed-Fed, a comprehensive and generic FL framework, comes in as a foundation for future practical FL system research. We also propose a novel resource-aware client selection algorithm to optimise the waiting time in the FL settings. We show that our approach can handle the straggler devices and dynamically set the training time for the selected devices in a round. Our evaluation has shown that the proposed approach significantly optimises waiting time in FL compared to conventional random client selection methods.**

I. Introduction

With the advances in hardware and software technologies, edge devices are becoming increasingly powerful and intelligent. This enables the researchers to bring machine intelligence from cloud-based data centres to edge devices such as mobile phones. For instance, Google Pixel's "Now Playing" option which allows us to recognise any song without internet and "recorder" app with on-device speaker diarization show how powerful the mobile phones are becoming. Federated learning (FL) [1] introduces an additional dimension in the machine learning community in which models are collaboratively trained on edge devices, with the data remaining local to the device, in contrast to the centralised training approach, and hence ensures the data privacy of the users. A general FL setting consists of a set of clients (edge devices) and a server. The server randomly chooses a subset of the devices from the available ones that want to take part in the FL round. A copy of the global model, which the server maintains, is first sent to this subset of clients. These clients use their data in the device to train models locally, and then send the updated weights back to the server. In conventional FL, the server cannot move on to the next step until it has received updates from every client. As a result, FL performance is constrained by the variability in waiting time experienced by each client due to model training on the device or communication delay during the transfer of model weights to the server. These slowdowns caused by the clients with weaker network connections or limited computation capabilities is known as "straggler effect". Once the server gets the updates from all the selected clients, the weights are aggregated using a strategy. This process is repeated for several rounds until the global model achieves the desired accuracy.

The random client selection approach for FL works well when there are no straggler devices. Prioritising for resource rich devices every time will result in the inability of straggler devices to participate in the FL process, and can lead to a loss of generalisation in the global model and fairness in the learning process. To address this challenge, a more sophisticated client selection algorithm is needed that considers the presence of stragglers while minimising waiting time. Hence the algorithm should aim to balance the participation of straggler devices with less waiting time and fairness in the FL process. Current FL frameworks face several challenges when it comes to deployment on edge devices such as mobile phones. These devices often have limited memory and computational resources, which makes it difficult to store and execute complex models. To overcome the existing challenges and make FL a practical and scalable solution for edge devices, it is important to develop FL frameworks that are designed specifically for these devices, taking into account their computational, privacy, and power constraints.

We provide a methodology for training the models on the device, along with an efficient client selection algorithm to handle straggler devices and FL functionalities, allowing them to be deployed on edge devices for FL settings. We also monitor the client devices' resources to ensure that they continue to function seamlessly even in dynamic environments. Since this work is related to client selection, we do not consider asynchronous federated learning approaches. In this paper, we focus on the use case of automatic speech recognition (ASR), demonstrating how our FL framework can be used to improve the accuracy and robustness of speech recognition models. We summarise our contributions as follows:

- We present Ed-Fed, an end-to-end FL framework for edge devices with a resource-aware time-optimised client selection algorithm.
- We provide a complete methodology for training entire or fine tuned models on mobile phones with support for

FL related functionalities.

- We formulate a client selection algorithm by considering the computation, storage, power, and phone-specific capabilities of the client devices with ability to handle stragglers, optimise the waiting time and adaptively assign the training time for devices based on the these information.
- We demonstrate the implementation with deployments on multiple mobile phones to quantify the waiting time in client selection.
- We present an extensive evaluation of our framework on both simulations and mobile phone settings using a custom created audio corpus with a heterogeneous set of speakers.

This paper is organised as follows. We provide a brief literature survey in Section II. Then, we discuss briefly about our Ed-Fed framework for FL settings in Section III. In Section IV, we discuss our proposed resource-aware client selection algorithm used in the server, followed by the framework evaluation in Section V and results in Section VI. Finally, concluding remarks are presented in Section VII.

II. Related works

FL Framework Recently, there has been a lot of interest in training ASR models in FL settings [2], [3], [4], [5], [6]. Implementing ASR for federated settings in mobile phones is a challenge in itself due to the complex model architectures and dynamic nature of the speech signals, as well as the limited resources available in mobile devices. Furthermore, due to factors such as noisy backgrounds, multiple accented speech, and different voice characteristics such as gender, pitch, phonation, loudness and tempo, the local speech data available on devices are non-IID in nature. This further complicates the training of ASR models. Several FL frameworks, including TensorFlow-Federated (TFF) [7], and LEAF [8], support only the simulation of FL systems and do not propose an edge device deployment. Flower framework [9], on the other hand, supports extending FL settings to edge devices. However, we can see that the ASR task with the flower framework is also limited to system-level simulations [2]. Moreover, they provide only transfer learning techniques [10] but not entire retraining of the model from scratch. To the best of our knowledge, no existing framework has provided a comprehensive implementation of ASR on-device training, as well as FL settings for mobile phones.

Client selection We focus mainly on bandit based client selection techniques in FL. In [11], the authors formulated the client selection as a traditional multi-arm bandit based problem to select clients with better quality of data to improve the FL accuracy. The authors proposed upper confidence bound based client selection in [12], [13] with the goal of reducing the overall time consumption of FL training including transmission time and local computation time. In [14], the authors intelligently select clients by exploiting the data correlations among clients to improve FL learning performance. However, none of these studies used actual computation resource information from the devices to explain training latency. Meanwhile, [15] took a similar approach to ours, taking into account resource information and grouping clients accordingly. Even though

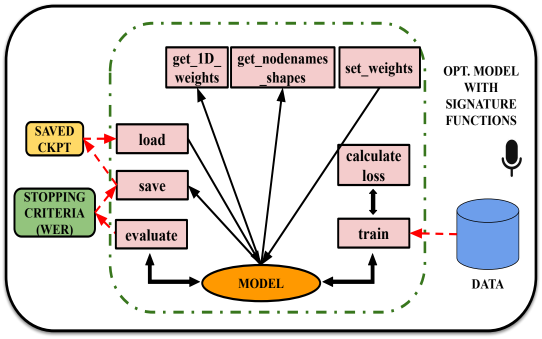


Fig. 1: Optimised model created for on-device training and FL setup

the clients with similar resources are selected together, the need for adaptive setting training epochs is needed to control the stragglers. Furthermore, they did not take the battery information into account, which is critical.

III. Ed-Fed framework

This section provides a detailed overview of our Ed-Fed framework. We first discuss the methodology for facilitating on-device training and weight updation of models in the clients, followed by a brief overview on the communication protocol and the server-side algorithms used.

A. Client

It is critical to develop a better methodology for converting a larger global model to a memory efficient and optimised version suitable for edge devices. We use TensorFlow [16] to create a well-optimised Flatbuffer format based conversion, which allows for significant model size reduction. We use the signature functions in Tensorflow to interface with the optimised model and perform operations such as training, inference, loading and saving the checkpoint weights, as described in [17], [18]. Furthermore, we build extra signature functions specifically for FL configuration to aid in efficient communication between clients and server. Figure 1 depicts the optimised model with eight signature functions that allow us to successfully train the model on the device, and perform FL related functions in the mobile devices. Along with the existing signature functions such as train, evaluate, save, load, and calculate_loss, we build three new signature functions:

- Get_1D_weights: reshapes an N-dimensional weight tensor from each node in the model graph to a 1-D array and returns a list of 1-D arrays.
- Get_nodenames_shapes: returns all the node names as well as the actual tensor shapes.
- Set_weights: reshape the aggregated 1-D weight array to N-D tensor and reloads it into the model.

One key functionality of these signatures is packing and unpacking the weights into 1-D and N-D arrays respectively. In [19], the authors explained that important sensitive data can be decoded if an intruder gets access to weights during communication. But if the weights are packed into a 1-D array, private information like shape of the weight matrices of each layer which can give useful insights about the model architecture and data patterns are hidden. These three main signatures, Get_1D_weights, Get_nodenames_shapes and Set_weights not only facilitate in implementing Ed-Fed framework but also induce privacy to the framework.

Due to the limited resources available in the edge devices, we prevent repeated storing of checkpoints after each epoch of training. Instead, we update a single checkpoint throughout the training. Hence, the optimised model (tflite) generated with the signature functions has all the functionalities which include support for training the models efficiently on the device as well as FL-based weight updation. The optimised model is utilised in our android application and our simulation systems, hence facilitating end-to-end real-time FL.

B. *Communication protocol*

We use the gRPC [20] communication protocol to ensure efficient communication between the server and clients. gRPC, like many RPC systems, is built on the concept of establishing a service, which describes the methods that can be called remotely with their input and return types. Protocol buffers are the most common interface definition language (IDL) used by gRPC to describe both the service interface and the message structure of the payloads. In our framework, three RPCs were used. “CommunicatedText” is the first, which is used to send the current context of the client and server. “GetGlobalWeights” is the second, which is used to send the current global FL weights to the clients who have been chosen for training. The final one is “GetFLWeights,” which is used to share the aggregated weights between the server and client.

C. *Server*

On the server side, there are two major components: the *Client selection with fairness* and *Aggregation strategy*. Client selection involves selecting k clients out of N available clients. In case of waiting time optimised client selection algorithms, the clients should be selected in such a way that the overall waiting time of the clients is minimised, while ensuring fairness in the selection of clients. More about the client selection algorithms is discussed in further sections.

The weight aggregation is an integral part of FL. The server strategy algorithms aggregates the weights (w_t^i) obtained from the selected k clients by choosing algorithms such as FedAvg [1], FedProx [21], etc., and the updated weights are sent back to the clients. Typically, in real-world settings the client’s local data will not be representative of the global data distribution because of the noisy background or the different voice characteristics. Hence, simply aggregating weights from such clients with low quality data will lead to a deviation from the global model. An improved solution is to generate a weighted aggregation of the models, with a weighting coefficient reflecting the quality of the model [2]. A client with a higher word error rate (WER) denotes poor performance of the global model. In such cases, we assign a lower weighting coefficient to that model during the aggregation. We use a weighted WER-based strategy in our Ed-Fed setting and is denoted as follows:

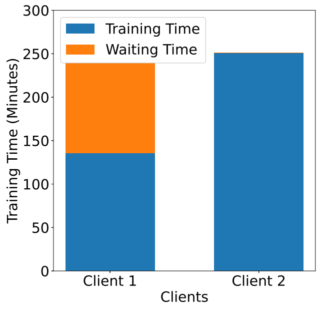


Fig. 2: Scenario 1: Slow vs Fast client.

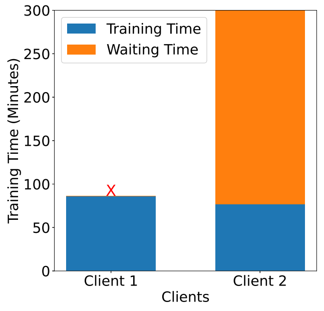


Fig. 3: Scenario 2: One client with insufficient battery life.

$$w_{t+1} \leftarrow \sum_{i=1}^{k} \alpha_i w_{t+1}^i \tag{1}$$

$$\alpha_i = \frac{\exp(1 - WER_i)}{\sum_{j=1}^{k} \exp(1 - WER_j)} \tag{2}$$

where the weights α_i’s are calculated using the softmax distribution obtained from the WER values from multiple clients.

Once clients are selected for training, the server notifies them to start training. The clients begin their training, and once they complete their training, they send their updated weight to the server. The server then aggregates them through a weight aggregation strategy. The aggregation strategy considers the uncertainties in the quality and quantity of data by using the WER based weighting used by each client obtained during the training process.

IV. Resource-aware time-optimised client selection

A. *Need for waiting time optimisation*

Waiting time is the amount of time the client waits for server to fulfill the request. This is mainly observed when the clients send their updated weights and request the aggregated weights to be returned by the server. Since client platforms have different training times, the server cannot compute the aggregated weights until responses are obtained from all clients, leading to higher waiting times for clients with faster computing capabilities.

To quantify the waiting time, we considered two common scenarios : (1) Select one fast client and one slow client and (2) Select one client with insufficient battery life made to run more number of epochs. Figure 2 shows client 1 having higher compute capabilities which has waited for a significant amount of time for the slower client 2 to finish its training process. Figure 3 presents the results of the experiment (Scenario 2). We can see that client 1 switches off in the middle of training, making client 2 wait for an infinite amount of time.

Considering all these problems, we created a resource-aware time-optimized client selection algorithm which takes the resources of the clients into consideration and assigns work adaptively to each client depending on that client’s resources. Our approach has three important phases, a resource informa-

Algorithm 1: NeuralUCB-m

```
Data: Number of training rounds $T$, exploration parameters
 $\{\alpha_t\}_{t \in T}$, Hidden layer size $m$, Context vectors at $t^{th}$
 round: $\{c_{t,s} | s \in \{1 \cdots N\}\}$
Initialisation: Randomly initialise $\boldsymbol{\theta_0^s}$,
 $\mathbf{Z}_{0,s} = \lambda \mathbf{I} \ \forall s \in 1 \cdots N$
for t=1 to $T$ do
    for s=1 to $N$ do
        $\left[\hat{b\_t}_{t,s}, \hat{d}_{t,s}\right] \leftarrow f_s(\mathbf{c_{t,s}}; \boldsymbol{\theta}_{t-1}^s)$
        $U_{t,s} \leftarrow -\hat{b\_t}_{t,s} +$
          $\alpha_t \sqrt{\nabla f_s(\mathbf{c_{t,s}}; \boldsymbol{\theta}_{t-1}^s)^T \mathbf{Z}_{t-1,s}^{-1} \nabla f_s(\mathbf{c_{t,s}}; \boldsymbol{\theta}_{t-1}^s)/m}$
    end
    $S_t \leftarrow$ Top $min(k, |U_t|)$ clients from $U_t$
    Run the clients $s \in S_t$ and get the true $[b\_t_{t,s}, d_{t,s}]$
    Update $\mathbf{Z}_{t,s} \leftarrow$
      $\mathbf{Z}_{t-1,s} + \nabla f_s(\mathbf{c_{t,s}}; \boldsymbol{\theta}_{t-1}^s)^T \nabla f_s(\mathbf{c_{t,s}}; \boldsymbol{\theta}_{t-1}^s)/m$
    $\mathbf{D}_{t,s} \leftarrow \mathbf{D}_{t-1,s} \cup (\mathbf{c_{t,s}}, [b\_t_{t,s}, d_{t,s}])$
    $\boldsymbol{\theta}_t^s \leftarrow$ TrainNN $\left(\mathbf{D}_{t,s}, \boldsymbol{\theta}_{t-1}^s\right)$
end
```

tion extraction, neural reward generation and a resource-aware time-optimised algorithm for clients.

B. Resource information

We aim to investigate the relationship between the resource availability and the fluctuations in the training time. We also aim to assess the impact of training on battery drain, which is crucial in real-world scenarios to prevent device shutdown during extended usage. The different resources which we considered for estimating the training time and battery drain of a particular phone are (1) memory-related information, (2) power-related information, (3) CPU-usage-related information (CI), and, (4) phone-specific information (PI). Memory-related information is captured using: (a) total RAM (TR), and, (b) available RAM (AR). In case of power-related information, (a) available battery charge (AC) and (b) battery status (BS) are used. For CI, the average usage of CPU across multiple cores is used. In order to find the PI, we used the Antutu score. The breakup of these resources into individual components is important because each of these can individually effect the training time. All these resources together form the context information of the client device, i.e., $c = [TR, AR, AC, BS, CI, PI]$. Obtaining context information from the phone before each FL round is important as time taken to complete the training process depends on it. We make use of this information by training a neural network to understand these dependencies and predict the expected training time and battery charge consumption based on a context vector. Once, we get to estimate these two parameters of each client, we can group clients together in a manner that minimises the waiting time of each client.

C. Neural reward generator

In this section, our goal is to learn the training time, and battery drop based on the given resources. Further, use these values to learn a better client selection by removing straggler clients or by doing an adaptive setting of the training time for clients. We will first give a brief overview of the basic contextual combinatorial multi-armed bandits (CC-MAB) problem using neural upper confidence bound (NeuralUCB)[22] policy. We will then use this in our FL framework setting to optimise the overall waiting time, based on the context information.

We formulate our client selection problem as a CC-MAB setting with N arms in which the agent interacts with the arms for T rounds. Initially, the true rewards generated by the arms are unknown, and the agent observes only the N context vectors from corresponding arms: $\{\mathbf{c_{t,s}} \in \mathbb{R}^d | s \in \{1 \cdots N\}\}$. Let $\mathbf{S} \subseteq 2^N$ be the set of all proper subsets of N available arms in the setting. At each round t, the agent predicts the rewards $\{\hat{r}_{t,s}\}_{s \in \mathbf{S_t}}$ using a reward estimating function (f) parameterised by $\boldsymbol{\theta}$, such that $\hat{r}_{t,s} = f(\mathbf{c_{t,s}}; \boldsymbol{\theta})$. Then, the agent selects a subset, $\mathbf{S_t} \in \mathbf{S}$ containing k arms based on the calculated rewards. In most cases, the agent is aware of the parametric form of the reward estimating function that is being used. In linear upper confidence bound problems (LinUCB)[23], for example, the reward is calculated as $\hat{r}_{t,s} = \boldsymbol{\theta_*}^T \mathbf{c_{t,s}}$. After playing the arms in $\mathbf{S_t}$, the agent observes the true rewards given as $\{r_{t,s}\}_{s \in \mathbf{S_t}}$. The goal of the agent is to maximise the expected reward, which is equivalent to minimising R_T, the cumulative regret over T rounds:

$$R_T = \mathbb{E}\left[\sum_{t=1}^{T}\left(\sum_{s \in \mathbf{S_t^*}} r_{t,s} - \sum_{s \in \mathbf{S_t}} \hat{r}_{t,s}\right)\right] \tag{3}$$

where $\mathbf{S_t^*}$ is the set of k optimal arms with maximum true rewards at round t.

Unlike the classical linear contextual bandit where the reward estimating function $f(\mathbf{c_{t,s}}; \boldsymbol{\theta})$ is linear, we utilises a neural network to deal with the intricate relationship between context features and rewards. The neural network based reward estimating function $f(\mathbf{c_{t,s}}; \boldsymbol{\theta})$ estimates the expected reward of an action based on past observations, and is parameterised as:

$$f(\mathbf{c_{t,s}}; \boldsymbol{\theta}) = \sqrt{m}\mathbf{W_L}\sigma\left(\mathbf{W_{L-1}}\sigma\left(\cdots\sigma\left(\mathbf{W_1}\left(\mathbf{c_{t,s}}\right)\right)\right)\right) \tag{4}$$

where L is the total number of hidden layers, m is the hidden layer size (assumed to be same for all the layers for convenience), σ is the activation function, and $\mathbf{W_l}$ corresponds to the weight matrix in l^{th} layer. Here $\boldsymbol{\theta}$ is the vectorised weight matrices from all the hidden layers in the neural network and is given as $\boldsymbol{\theta} = \left[\text{vec}(\mathbf{W_1}^T), \text{vec}(\mathbf{W_2}^T) \cdots \text{vec}(\mathbf{W_L}^T)\right]$.

Our proposed approach for client selection considers the clients or edge devices participating in FL as the arms with resource information discussed in Section IV-B as the context vectors. We also observe that using a single reward generation model $f(\mathbf{c_{t,s}}; \boldsymbol{\theta}_{t-1})$ for multiple client devices (NeuralUCB-s) may lead to performance degradation if the edge devices have different intrinsic characteristics such as age and usage history. For example, consider two identical phones, one of which has been in use for 5 years and the other of which is brand new. However, if we calculate the training time and battery drop of both phones under similar contexts, they do not match. We see that the older phone performs badly and

Algorithm 2: Resource-aware time-optimised algorithm for client selection

```
Data: Context vectors of clients at $t^{th}$ round: $\{c_{t,s} | s \in \{1 \cdots N\}\}$
      Number of samples at round t: $(n_{t,1}, n_{t,2}.., n_{t,N})$,
      Number of clients to be selected: $k$, Minimum
      and maximum number of epochs to run in a FL
      round: $(e_{min}, e_{max})$, Batch size: $bs$, Battery
      threshold $\gamma$.
Result: k clients along with their respective $e_{t,i}$
for i=1 to N do
    $\hat{b\_t}_{t,i},\ \hat{d}_{t,i} \leftarrow f_i(\mathbf{c_{t,i}}; \boldsymbol{\theta}^i_{t-1})$ ;        /* Step 1 */
    $b_{max_{t,i}} \leftarrow \lfloor (AC_{t,i} - \gamma)/\hat{d}_{t,i} \rfloor$;
    $e_{max_{t,i}} \leftarrow min(e_{max}, \lfloor (b_{max_{t,i}}/(n_{t,i}/bs)) \rfloor)$ ;
     /* Step 2 */
    if $e_{max_{t,i}} \geq e_{min}$ then        /* Step 3 */
        $P_t.append(c_{t,i})$;
    end
end
$S_t \leftarrow$ NeuralUCB-m($P_t$);        /* Step 4 */
$m_t \leftarrow 0$;
foreach client $i$ in $S_t$ do
    $m_t \leftarrow min(m_t, e_{max_{t,i}} * (n_{t,i}/bs) * \hat{b\_t}_{t,i})$ ;
     /* Step 5 */
end
foreach client $i$ in $S_t$ do
    $e_{t,i} \leftarrow \lfloor (m_t/\hat{b\_t}_{t,i}) * (bs/n_{t,i}) \rfloor$ ;    /* Step 6 */
end
```

drains faster due to the aging of batteries. Also, it is dependent on how extensively the phone is used over time. If we use a single model, we miss out on these relationships, and the model gets confused if such clients exist in our rounds with degradation in performance. On the other hand, personalised reward generation models $f_s(\mathbf{c_{t,s}}; \boldsymbol{\theta}^s_{t-1})$ (NeuralUCB-m), can better adapt to the unique characteristics of each client device and provide more accurate results. In our approach, the neural network predicts the training time per batch of samples and also the drop in battery percentage given the resource information as the context vector, i.e., $\left[\hat{b\_t}_{t,s}, \hat{d}_{t,s}\right] = f_s(\mathbf{c_{t,s}}; \boldsymbol{\theta}^s_{t-1})$. We use the negative of training time per batch $(-\hat{b\_t}_{t,s})$ for each arm as the reward in the UCB setting and the battery drop is used for straggler handling. Once the training is completed for round t, the clients will send their $[b\_t_{t,s}, d_{t,s}]$ along with the updated weights. Then, $(\mathbf{c_{t,s}}, [b\_t_{t,s}, d_{t,s}])$ will be added to the corresponding clients' dataset $(\mathbf{D}_{t,s})$ and will be used for training the neural network.

The detailed algorithm for neural combinatorial contextual bandits is shown in Algorithm 1.

D. Resource-aware time-optimised algorithm for client selection

Initially, all the available clients $\{s \in \{1 \cdots N\}\}$ who wish to participate in t^{th} FL round, express their interest by sending their context vectors $\{c_{t,s}\}$ and number of samples available for training $(n_{t,i})$ to the server. The server then selects k clients based on Algorithms 1 and 2 and notifies the selected clients about their selection along with the number of epochs $(e_{t,i})$ they need to run in that round which is computed using Algorithm 2. The methodology is as follows:

Step 1 Calculate: (a) time taken to complete training a batch of data $(\hat{b\_t}_{t,i})$, (b) drop in battery on training a batch of data $(\hat{d}_{t,i})$, and (c) the maximum number of batches $(b_{max_{t,i}})$, each client can run while maintaining the battery above $\gamma\%$.

Step 2 Calculate the maximum number of epochs $(e_{max_{t,i}})$ each client can run from $b_{max_{t,i}}$, $n_{t,i}$, e_{max} and bs

Step 3 Filter out the clients who can run a minimum of e_{min} epochs into a set (P_t)

Step 4 Create a set (S_t) by picking $min(k, |P_t|)$ clients of P_t using Algorithm 1.

Step 5 Calculate the time taken to run $e_{max_{t,i}}$ epochs for each client and take the minimum of all these times (m_t). m_t will be the maximum time the FL round can happen while ensuring that no client switches off in this time.

Step 6 Calculate the number of epochs $(e_{t,i})$ each client can run in m_t time.

Step 7 Notify each client of S_t about their selection for that FL round along with their respective $e_{t,i}$.

Thus, this algorithm ensures to adaptively choose the number of training epochs for the chosen clients while taking into account the available resources and minimising the waiting time for each client during FL rounds.

V. Ed-Fed framework setup

We explain the entire setup used for evaluating Ed-Fed framework in both simulation and mobile environments.

A. Setup for system-based evaluation

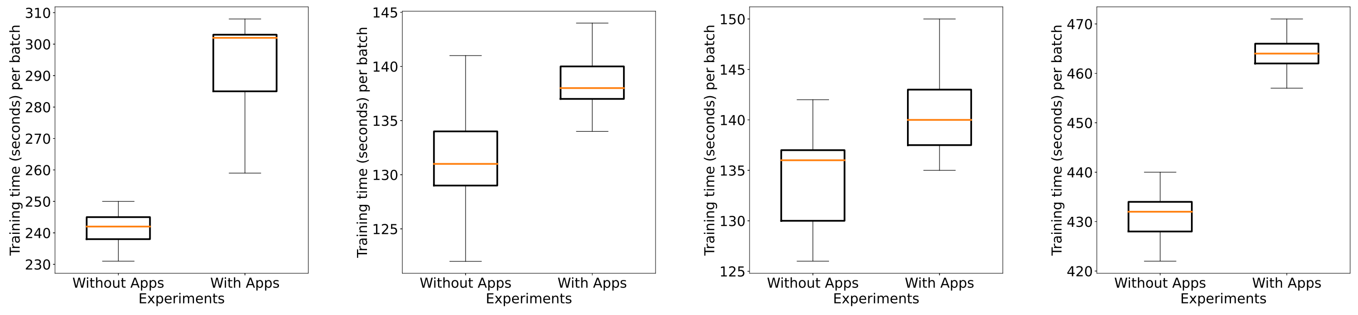
In this section, we present the simulation settings used for evaluating our Ed-Fed framework for ASR tasks. The objective of this experiment to make the global model robust to multiple accents by learning from different accented clients.

We use an end-to-end acoustic model similar to DeepSpeech2 [24] architecture, collectively trained on datasets such as Librispeech [25], commonvoice [26] and tedlium [27] as our initial global model. For FL experiments, we created an audio corpus using a text-to-speech (TTS) system [28] for 15 different accented speakers to simulate unique clients. Each speech sample is about 8-10 seconds, with an average label length limited to 150 characters. We run the FL experiments by associating one speaker data to one client. To evaluate the global model's accuracy, we created a test set consisting of speech samples from various speakers.

The simulation environment uses a single server and multiple python clients. We conduct a series of experiments, where we use our Ed-Fed framework to train the baseline ASR model on our audio corpus. We repeated the experiment for different values of k varying from 3 to 5. Each experiment is run for $T = 5$ rounds with a fixed k and the k clients are randomly chosen from a pool of 10 readily available clients. We use the server strategy algorithm explained in Section III-C to

TABLE I: Details about mobile phone hardware.

Device Model	RAM	CPU	OS	SoC
OnePlus 7T-1	4.0|8 GB	Octa-core Max 2.96 GHz	Oxygen 11 (Android 11)	Snapdragon 855 Plus
OnePlus 7T-2	4.1|8 GB	Octa-core Max 2.96 GHz	Oxygen 11 (Android 11)	Snapdragon 855 Plus
OnePlus 5T	3.6|6 GB	Octa-core Max 2.45 GHz	Oxygen 9 (Android 9)	Snapdragon 835
Xiaomi 11 Pro	4.8|8 GB	Octa-core Max 2.05 GHz	MIUI 13 (Android 11)	Helio G96



(a) OnePlus 5T (b) OnePlus 7T - 1 (c) OnePlus 7T - 2 (d) Xiaomi 11 Pro

Fig. 4: Effect of RAM on training time

aggregate the weights from the selected k clients. We train the model with 25 samples and a validation set of 10 samples. All of our experiments were carried out using NVIDIA RTX 3090 and 3080 GPUs on a 10-core Intel i9-10900K CPU.

B. Mobile phone based Evaluation

With this set of experiments, we discuss the technical details associated with running Ed-Fed on edge devices. We present results from deploying our framework on our custom-built android application which allows the user to record speech samples. These recorded samples gets stored into the local memory of the application.

We save the datasets for training and testing in the storage cache of the mobile phone. We train the model with 25 samples and a validation set of 10 samples. We host our python Ed-Fed server with WER based aggregation strategy on a local machine. The Ed-Fed clients are the Android mobile phones listed in Table I. We use the optimised model mentioned in Section III-A for the on-device personalisation of the ASR model.

VI. Results

A. Effect of resources on the training time

Figure 4 depicts the results of an experiment to show the effect of varying RAM on training time with the help of two scenarios: one with background apps running alongside our FL android application (less AR) and other with no background apps (high AR). We observe that with decrease in available RAM, there is a significant increase in training time per batch, across all the mobile phones. This is especially noticeable in Figure 4a and Figure 4d, where we see a jump of 49 and 33 seconds in training time respectively. Figure 5 presents the results obtained during the experiment conducted to check the effect of the battery percentage on the phone's training time.

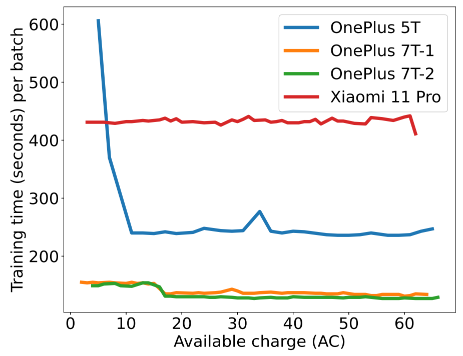
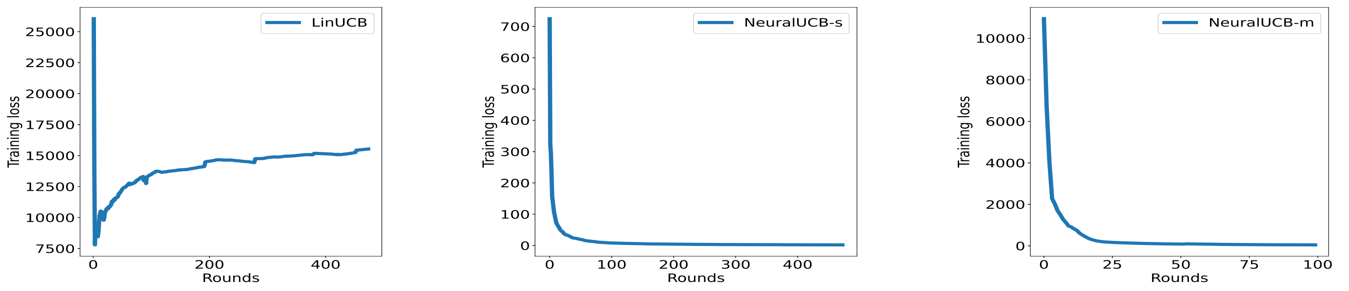


Fig. 5: Battery vs training time

We can infer from the figures that training time shoots up abruptly when in lower battery bands ($\gamma = 20\%$), whereas it is almost constant in the upper battery bands for all the phones except Xiaomi 11 Pro. This is particularly evident in the OnePlus 5T phone where training time increased 2.4 times the regular time in lower battery bands.

B. Neural reward generator

For our contextual bandits experiment, we chose $N = 4$ clients and the number of rounds $T = 475$. We ran multiple iterations of on-device training on the N mobile devices to generate the context vectors containing the resource information and noted down the time taken per batch and the battery drop. We use the experimental setup described in [23] for LinUCB. The neural networks used in the NeuralUCB-s and NeuralUCB-m share the same architecture, consisting of a simple fully connected feedforward network with two hidden layers of 32 and 16 units, respectively with ReLu activations. The input to the network is a d dimensional context vector, and the outputs are the time and battery drop. For



(a) LinUCB (b) NeuralUCB single model (c) NeuralUCB multiple models

Fig. 6: Training loss vs FL rounds for different reward generator functions

TABLE II: Client selection evaluation

Approach	Experiments	Clients	AC	BS	e_{min}	e_{max}	$b\hat{\_}t_{t,i}$	$b_{max_{t,i}}$	$e_{max_{t,i}}$	m_t (minutes)	$e_{t,i}$	Actual time per batch (seconds)	Waiting time (minutes)
Our Client Selection	Scenario 1	Client 1	100	1	1	7	431.93	46	7	146.57	4	430	7.42
		Client 2	100	1	1	7	251.25	46	7		7	233	
	Scenario 2	Client 1	60	0	1	7	251.25	18	3	51.86	3	233	14.25
		Client 2	100	0	1	7	130.36	50	7		4	132	
Random Client Selection	Scenario 1	Client 1	X								7	430	114.92
		Client 2									7	233	
	Scenario 2	Client 1									7	233	∞
		Client 2									7	132	

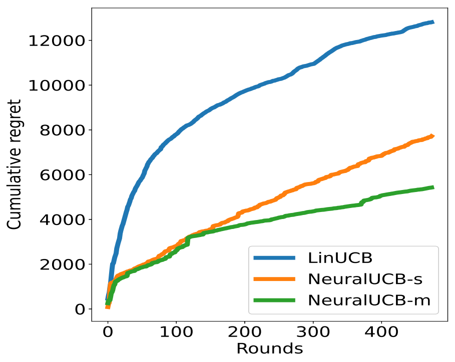


Fig. 7: Comparison of UCB based client selection algorithms

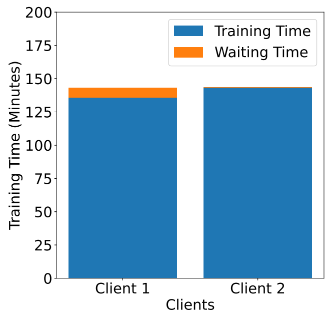


Fig. 8: Scenario 1: Slow vs Fast client.

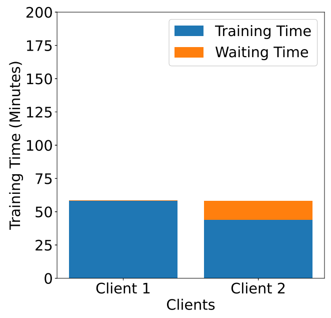


Fig. 9: Scenario 2: One client with insufficient battery life.

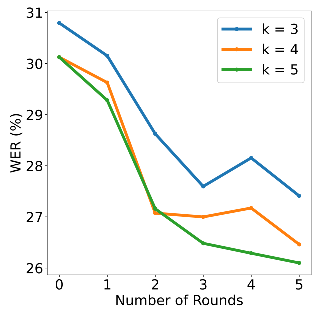


Fig. 10: Trend of WER for various values of k

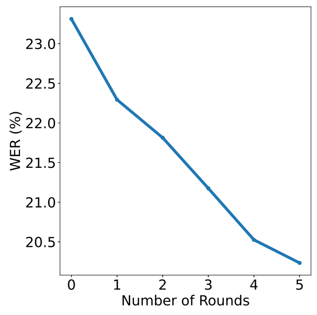


Fig. 11: Performance of Ed-Fed on mobile phones

NeuralUCB-s and LinUCB, we use a single reward generating function for all the clients. As a result the context vector $c = [TR, AR, AC, BS, CI, PI]$ contains all six features as discussed in Section IV-B. In contrast, we do not need to provide the total RAM (TR) and phone-specific information (PI) as features in the NeuralUCB-m approach with personalised models for each client device. Hence, $d = 4$ is used in this case. We do a grid search over $\{0.01, 0.1, 1.0, 10\}$ for the exploration term multiplier $\alpha_t = \alpha$ for all the experiments, and tuned the parameter in such a way that the fairness achieved is similar across all the algorithms. For LinUCB, we selected $\alpha = 10.0$ and a value of 0.01 for NeuralUCB based algorithms.

Figure 6 trace the model's mean square error loss over rounds. We can see that neural network-based algorithms outperform LinUCB. Figure 6a demonstrates that linear models are unable to accurately estimate the output and learn feature representations from the data. Comparing the results of NeuralUCB-s and NeuralUCB-m from Figures 6b and 6c respectively, we can observe that both loss curves looks similar with NeuralUCB-m performing slightly better over a long run. Figure 7 plots the overall regret of all the algorithms against rounds. We present the findings based on an average of five repeated experiments using various random dataset shuffles. Further, we can also infer from Figure 7 that NeuralUCB-m with disjoint personalised models for each client appears to outperform all other algorithms.

C. Resource-aware time-optimised client selection

Figure 8 and Figure 9 are the results obtained when we redo the experiments of Scenario 1 and Scenario 2 of Section IV-A with our client selection algorithm using our neural reward generator at $t = 476$. Table II contains the detailed information of these two experiments. In Scenario 1, our algorithm identified that client 1 has weaker computing capabilities based on the $\hat{b\_t}_{t,i}$ values computed. Since, client 1 has weaker computing capabilities, our algorithm assigns a smaller $e_{t,1}$ of 4 epochs in contrast to the 7 epochs given by random client selection approach. Thereby, reducing the overall waiting time to 7.42 minutes when compared to random client selection's 114.92 minutes. In Scenario 2, it is clear from $e_{max_{t,i}}$ values of the Table II that client 1 has weak battery resources. Our client selection algorithm deduced that client 1 cannot run 7 epochs and prevented the whole FL round from stopping by assigning a smaller $e_{t,1}$ of 3 epochs unlike random client selection. Further, our algorithm adjusted $e_{t,2}$ with respect to $e_{t,1}$ by giving it a value of 4 and reduced the waiting time even more. Whereas, the random client selection without any knowledge of resource information asks the client 1 to run e_{max} epochs. This leads to power shutdown of client 1, thereby making client 2 wait for infinite amount of time.

D. Ed-Fed framework evaluation

Figure 10 displays the results obtained by choosing various values of k for each FL experiment. The figure shows three line plots, each corresponding to different values of k. Each line plot shows how the global model performs on the global test set after each round of FL. The global test set consists of 15 unseen speech samples each from 10 speakers with 4 different accents. We can deduce from the figure that as k increases, the WER of the global model decreases.

Figure 11 depicts the findings obtained on deployment of our Ed-Fed framework on multiple phones. The experiment is carried for 5 rounds on 4 mobile devices. In each round, 2 clients are selected. The round 0 in the figure refers to the initial global weights. All the checkpoints that are obtained at the end of each FL round are put to the test on a global test set. As could be predicted, the WER declines as the number of FL rounds grow.

VII. CONCLUSIONS AND FUTURE DIRECTIONS

In this work, we present Ed-Fed, a first-of-its-kind end-to-end federated learning framework that will serve as a foundation for future research in practical FL systems. We also propose a client selection algorithm that takes into account factors such as computation, storage, power, and device-specific capabilities to handle stragglers, optimise waiting time, and adapt training time based on resource information. The framework has been thoroughly tested in simulations and on actual edge devices, and in the future, we plan to incorporate communication latency parameters to measure waiting time.

REFERENCES

[1] McMahan, B., et al. "Communication-efficient learning of deep networks from decentralized data", Artificial intelligence and statistics. PMLR, 2017.

[2] Yan, G. et al. "End-to-end speech recognition from federated acoustic models." ICASSP 2022-2022 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2022.

[3] Cui, X., Songtao L., and Brian K., "Federated acoustic modeling for automatic speech recognition.", IEEE Int. Conf. on Acoustics, Speech and Signal processing (ICASSP), 2021.

[4] Yu, W., Freiwald, J., Tewes, S., Huennemeyer, F. and Kolossa, D.," Federated learning in ASR: Not as easy as you think", In Speech Communication, 14th ITG Conference (pp. 1-5), 2021.

[5] Guliani, D., Beaufays, F. and Motta, G.,"Training speech recognition models with federated learning: A quality/cost framework", In IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), 2021.

[6] Dimitrios D., Kenichi K., Robert G.,Yashesh G., and Eskimez, S E., "A federatedapproach in training acoustic models," inProc. Interspeech, 2020.

[7] Google. "Tensorflow federated: Machine learning on decentralized data.", https://www.tensorflow.org/federated, 2020. accessed 25-Mar-20.

[8] Caldas, S., Duddu, S. M. K., Wu, P., Li, T., Konecn y, J., McMahan, H. B., Smith, V., and Talwalkar, A., "Leaf:A benchmark for federated settings", arXiv preprint arXiv:1812.01097, 2018.

[9] Beutel, D. J., et al. "Flower: A friendly federated learning research framework", arXiv preprint arXiv:2007.14390 (2020).

[10] Mathur, A., Beutel, D. J., de Gusmao, P. P. B., Fernandez-Marques, J., Topal, T., Qiu, X., ... & Lane, N. D. (2021). On-device federated learning with flower. arXiv preprint arXiv:2104.03042.

[11] Lai, Fan, et al. "Oort: Efficient Federated Learning via Guided Participant Selection." OSDI. 2021.

[12] Xia, Wenchao, et al. "Multi-armed bandit-based client scheduling for federated learning.", IEEE Transactions on Wireless Communications (2020).

[13] Yoshida, N., et al. "Mab-based client selection for federated learning with uncertain resources in mobile networks." 2020 IEEE Globecom Workshops.

[14] Cao, Hangrui, et al. "Birds of a Feather Help: Context-aware Client Selection for Federated Learning." Int. Workshop on Trustable, Verifiable and Auditable Federated Learning in Conjunction with AAAI (FL-AAAI). 2022.

[15] Banerjee, S., Vu, X. S., Bhuyan, M.,"Optimized and Adaptive Federated Learning for Straggler-Resilient Device Selection", In 2022 International Joint Conference on Neural Networks (IJCNN) (pp. 1-9).

[16] Abadi, M. et.al., "TensorFlow: a system for Large-Scale machine learning", In 12th USENIX symposium on operating systems design and implementation.

[17] Tflite, "On-device model personalization", https://blog.tensorflow.org/2019/12/example -on-device-model-personalization.html, 2020.

[18] Sasindran, Z., Suresh, R.R., Rao, P. and Prabhakar, T.V., "Training end-to-end speech-to-text models on mobile phones.", arXiv preprint arXiv:2112.03871,2021.

[19] Carlini, N., Liu, C. and Kos, J. ,Erlingsson, U., and Song, D., "The secret sharer: Measuring unintended neural networkmemorization & extracting secrets", URL http://arxiv.org/abs/1802.08232.

[20] Foundation, C. N. C, " grpc: A high performance, opensource universal rpc framework", URL https://grpc.io. Accessed: 2020-03-25.

[21] Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A. and Smith, V., "Federated optimization in heterogeneous networks", Proceedings of Machine Learning and Systems, pp.429-450, 2020.

[22] Zhou, D., Li, L., and Gu, Q, "Neural contextual bandits with ucb-based exploration" In International Conference on Machine Learning, 2020.

[23] Li, L., Chu, W., Langford, J., and Schapire, R. E." A contextual-bandit approach to personalized news article recommendation", In Int. Conf. on World wide web.

[24] Amodei, D., et al. "Deep speech 2: End-to-end speech recognition in english and mandarin." Int. Conf. on machine learning, PMLR, 2016.

[25] Panayotov, V., Chen, G., Povey, D. and Khudanpur, S.," Librispeech: an asr corpus based on public domain audio books", IEEE Int. Conf. on Acoustics, Speech and Signal processing (ICASSP), 2015.

[26] Ardila, R., et al. "Common voice: A massively-multilingual speech corpus." arXiv preprint arXiv:1912.06670 (2019).

[27] Hernandez, F. et. al..," TED-LIUM 3: twice as much data and corpus repartition for experiments on speaker adaptation", In Int. Conf. on Speech and computer, 2018.

[28] Flood, J., "NaturalReader: A New Generation Text Reader",. Developmental Disabilities Bulletin, 35, pp.44-55, 2017.