Enhancing Efficiency in Multidevice Federated Learning through Data Selection

FAN MO, Imperial College London, UK (work done at Nokia Bell Labs)
MOHAMMAD MALEKZADEH, Nokia Bell Labs, UK
SOUMYAJIT CHATTERJEE, Nokia Bell Labs, UK
FAHIM KAWSAR, Nokia Bell Labs & University of Glasgow, Scotland, UK
AKHIL MATHUR, Nokia Bell Labs, UK

Federated learning (FL) in multidevice environments creates new opportunities to learn from a vast and diverse amount of private data. Although personal devices capture valuable data, their memory, computing, connectivity, and battery resources are often limited. Since deep neural networks (DNNs) are the typical machine learning models employed in FL, there are demands for integrating ubiquitous constrained devices into the training process of DNNs. In this paper, we develop an FL framework to incorporate on-device data selection on such constrained devices, which allows partition-based training of a DNN through collaboration between constrained devices and resourceful devices of the same client. Evaluations on five benchmark DNNs and six benchmark datasets across different modalities show that, on average, our framework achieves \( \sim 19\% \) higher accuracy and \( \sim 58\% \) lower latency; compared to the baseline FL without our implemented strategies. We demonstrate the effectiveness of our FL framework when dealing with imbalanced data, client participation heterogeneity, and various mobility patterns. As a benchmark for the community, our code is available at github.com/dr-bell/data-centric-federated-learning.

1 INTRODUCTION

Multidevice FL enables learning new insights from various clients by training DNNs on personal data in a more private manner [23, 45]. However, some wearable or embedded devices have limited computing resources, little on-device storage, and variable network connectivity. Most existing FL frameworks do not account for constraints of personal devices and instead assume either modern smartphones [18, 48] or accelerators [9, 63] as target client devices. Although smartphones and accelerators are less powerful than cloud machines, they are nevertheless equipped with mobile GPUs, possess gigabytes of runtime memory, and have fairly stable connectivity to a central server, which simplifies the requirements of DNN training [35, 61].

Authors’ addresses: Fan Mo, mofan1992@gmail.com, Imperial College London, UK (work done at Nokia Bell Labs); Mohammad Malekzadeh, mohammad.malekzadeh@nokia-bell-labs.com, Nokia Bell Labs, UK; Soumyajit Chatterjee, soumyajit.chatterjee@nokia-bell-labs.com, Nokia Bell Labs, UK; Fahim Kawsar, fahim.kawsar@nokia-bell-labs.com, Nokia Bell Labs & University of Glasgow, Scotland, UK; Akhil Mathur, akhilmathurs@gmail.com, Nokia Bell Labs, UK.
**Challenge.** We focus on multidevice FL for *ubiquitous constrained devices* (UCDs). Examples are wearable devices such as earbuds, glasses, or rings, and embedded devices such as environmental or industrial sensors. We are driven by the motivation that UCDs are becoming the primary data-producing devices for individuals and industries due to various on-device sensors, such as cameras and inertial measurement units. UCDs offer a wealth of spatio-temporal data that is frequently absent when focusing solely on smartphones or plugged-in devices. For instance, people often opt for wristbands, smart rings, or earbuds over carrying a smartphone in their hand or pocket while engaging in outdoor, industrial, or sports activities [47, 52]. Moreover, low-cost sensors distributed across farms or forests remotely monitor crops and livestock and detect potential threats [49]. Such abundance and diversity in sensory data captured by UCDs can facilitate the learning of more effective models for emerging applications in personal and industrial environments. Nevertheless, there are two challenges for FL on UCDs:

The first challenge is the network connectivity to a central server. Usually, UCDs use an *access point* (AP) over wireless channels, e.g., a smartphone or a router, as a relay to connect to the central server. Such wireless connections, e.g., via Bluetooth Low Energy (BLE), are unreliable due to the devices’ mobility. Particularly for outdoor settings, UCDs may not always be connected to the companion AP, and hence therefore no immediate connection to the central server.

The second challenge emerges from constraints on the memory and compute resources available on UCDs, particularly the model size and functionality of DNNs that can be trained locally on UCDs. Although a naive solution is to transfer all data from UCDs to their respective APs and perform training of DNNs on APs, storing all data could exceed the storage capacity of UCDs, especially when UCDs disconnect from their AP often. To tackle these challenges effectively, we require FL frameworks that dynamically distribute data and computations between UCDs and APs.

**Solution.** We propose an end-to-end FL framework that orchestrates the local training of DNNs among UCDs and APs by integrating on-device data selection alongside partition-based model training. We refer to our proposed framework as Centaur. (A) Considering computation constraints, Centaur initializes the DNN in two partitions: an encoder (i.e., feature extractor) to be only trained on APs, followed by a lightweight classifier to be trained on both APs and UCDs (§3.2).

This is inspired by split [17, 55] and transfer [58] learning, where UCDs keep the encoder frozen and only train the classifier, while the AP train the full DNN. (B) Considering memory and connectivity constraints, Centaur performs data selection [22] by analyzing the training loss and the gradients norm of the classifier part, to decide which data points captured by UCDs contribute more to the training of which part of the DNN. Here, data points are categorized as either of (i) discarded if they have very low loss values, (ii) kept locally on the UCD to train the classifier part if their loss values or gradients norm are not very high, and (iii) transmitted to the AP to train both encoder and classifier part if they cause high values for both loss and gradients norm\(^1\) (§3.3).

---

\(^1\) The assumption is each pair of UCD and AP belongs to the same client, thus transmitting data from a UCD to its AP does not violate the clients’ privacy.
Centaur differs from the prior works on hierarchical FL [1, 36], since they neither perform partition-based training nor incorporate data selection. This makes our FL framework adaptive to the available resources in a multidevice setting.

**Evaluation.** We perform evaluations on five benchmark DNNs and six benchmark datasets. We show that compared to existing FL alternatives, Centaur not only saves more bandwidth by reducing the communication cost associated with offloading samples from the UCD to the AP, but also reduces the latency of training, and as a result, the energy consumption, on both the AP and the UCD. As a prime example of our experimental results, Figure 1 shows the latency vs. accuracy of performing FL for two different tasks: (1) four benchmark DNNs, with their encoder pre-trained on ImageNet [12], and then finally trained for CIFAR10 [26] image classification with 100 clients, and (2) a benchmark ConvNet [8], trained from scratch, on three human-activity recognition (HAR) datasets with 20 clients. In both figures, we compare Centaur with the baseline of standard FL on UCDs without our implemented strategies. Centaur, through data selection and partition-based model training, achieves up to 19% higher accuracy and 58% lower latency, on average. We provide thorough experiments and comparisons with other alternative baselines in §6.

**Contribution.** Our solution enables UCDs to efficiently participate in multidevice FL alongside their upstream APs, by incorporating a customizable data selection scheme and a partition-based training regime. The impact is a decrease in computational and communication costs, as well as an improvement in the model’s test accuracy. Our experiments on a small testbed of Raspberry Pis validate that Centaur is promising in decreasing overall storage requirements and training time due to employing effective data selection. Moreover, our analysis on the mobility of UCDs accounts for the real-world spatiotemporal mobility of edge devices participating in FL. Our extensive empirical analyses, including measurement of cost, imbalance data, participation heterogeneity, and connection probability, indicate that Centaur achieves higher efficiency (i.e., high accuracy and low cost) across most scenarios. Our code is open-source to encourage further advancements in this domain.
2 IDENTIFYING DESIGN CHALLENGES: A MOTIVATIONAL STUDY

Terminology. A ubiquitous constrained device (UCD) is a mobile device with restricted hardware and connectivity resources. An access point (AP) is a resourceful device that has much better hardware and connectivity capabilities than its corresponding UCD. A server is a central entity owned by a service provider that organizes FL over several clients. An encoder ($E$) is the first part of the DNN and extracts the features of input data. A classifier ($C$) is the second part of the DNN and classifies the encoded features into a final prediction. Let $X$ denote the input data and $y$ denote its label.

2.1 The Study

We perform measurement experiments to identify the key design requirements that can facilitate the deployment of FL on constrained devices.

Setup. We create a testbed with four RaspberryPi 4 Model B as the clients [14, 56]; all running the 64-bit Raspbian OS with total secondary storage of 32GB. We set up a cohort of clients having varying primary memories, with clients C1, C2, C3, and C4 having 1GB, 2GB, 4GB, and 2GB of primary memory, respectively. For analyzing the CPU and memory consumption of the clients, we use the logs from the `/proc/stat` and `free` system calls, respectively. For the model architecture, we use a benchmark architecture MobileNetV3 [19] (pre-trained on ImageNet) as the encoder followed by a medium-sized classifier with two fully connected layers with an intermediate drop-out layer. We use CIFAR10 dataset and measure the resource and time consumed on-device for both training, per each epoch, and inference. Finally, for implementing FL setup, we use Flower [4] over four RaspberryPi devices and a Mac Mini as the server, as shown in Figure 2a.

Running Standard FL. We implement standard FL (for 100 rounds) over four RaspberryPi clients with the standard FedAvg [44]. We consider two scenarios: (1) when only the classifier part is trained, and (2) when the entire model (encoder and classifier) is trained. For the first scenario, we observed in our experiments that the standard FL runs successfully over all four clients. However, for the second scenario, client C1 with 1GB of primary memory fails to participate in the FL process. Notably, 1GB of primary memory is still significant in terms of memory capability and typical UCDs usually have a much lesser amount of memory than this. Therefore, it is clear that in such cases, it is much more challenging to deploy standard FL over UCDs. To further assess this, we conduct a series of experiments in the following section to investigate the resource and time consumption for training DNNs on our testbed.

On-device Training. Considering the challenges discussed previously, we analyze the latency and resource consumption for local training of MobileNetV3. Figure 2 shows that there is a significant rise in the overall CPU and memory consumption when the entire model (encoder and classifier) is trained on-device in comparison to when only the classifier is trained and the encoder kept frozen. A key observation is that by training only the classifier, we require less processing power and approximately the same amount of memory resources as when the entire model is used solely for inference. Moreover, Figure 3 shows that the
time taken to train the classifier is less than the time to perform a single forward pass on the entire model. These measurements confirm that training the classifier only is $50\times$ time efficient compared to training the entire model, which helps in reducing the battery consumption significantly. When we follow a naive approach of randomly dropping samples from the training dataset, we observe a decline in the overall training time of the classifier (see Figure 3b). However, this comes at the cost of reduced accuracy, which might be detrimental to the overall system.

2.2 Insights

Here, we summarize the understandings derived from our motivational study:

(1) **Resource and Training Efficiency.** As UCDs typically have limited memory, training on the device should be memory efficient. This motivates using a frozen pre-trained encoder, and only training the classifier part on the device [43]. However, freezing the weights of the encoder limits the expressiveness of the model and prevents the model from learning important features of the newly collected data over the UCDs. Thus, while minimizing the on-device model training on the UCDs, our system should ensure that the encoder is periodically trained on a portion of new data collected from the UCDs, potentially on a more resourceful device where the entire model can be trained.

(2) **Data Efficiency and Spatiotemporal Coverage.** Typically, UCDs are deployed to maximize the spatiotemporal coverage of activities and events. Hence, our design should leverage the spatiotemporal richness of such data to train a better model. Such spatiotemporal diversity also makes UCDs more prone to intermittent connectivity, which may require the framework to store training data on-device. UCDs possess limited secondary storage, and even when they have persistent connectivity to the AP, streaming high volumes of data could drain their battery. Thus, our design should dynamically select a small subset of data points for storing locally or transmitting to the APs. This needs to be designed in a sophisticated manner such that it does not compromise the overall performance of the system.
3 OUR PROPOSED FRAMEWORK: CENTAUR

With our understanding developed from the motivational study, we introduce our proposed framework. Figure 4 gives a schematic overview, and below we elaborate on the algorithms used in Centaur.

3.1 The Overview

The first level includes UCDs, which are connected to their APs via BLE, Wi-Fi, Backscatter, or other wireless technologies. We assume the connection between UCD and AP is not always available or reliable due to the natural mobility of UCDs. We consider that the data selection procedure and the training of the classifier only happen on UCDs. The second level, which includes APs, is responsible for training the encoder in conjunction with the classifier atop it. The third level includes a central server, which performs model aggregation for both classifiers and encoders that are trained by UCDs and APs. Here, we present a step-by-step description of our proposed framework.

Step ①: We assume that the server receives the information about UCDs’ resource. Accordingly, the server initializes a DNN model \( M \) that is formed of an encoder \( E \) and a classifier \( C \), and then distributes the \( E \) and \( C \) to the APs and subsequently to UCDs. We explain model initialization details in §3.2.

Step ②: UCDs start FL by performing data selection on the collected data \( D \), based on the computed loss value via the forward pass and the magnitude of the gradient of the classifier’s last layer. Each sample point will be categorized in one of (i) \( D_N \) to be discarded, (ii) \( D_C \) for locally training \( C \) on the UCD, or (iii) \( D_M \) to be transmitted to AP for training \( M \). We explain data selection details in §3.3.

Step ③: The UCD discards data \( D_N \) in the current epoch without any further training steps or backward pass on them.
Fig. 4. The overview of our framework, including model initialization (§ 3.2), data selection (§ 3.3), and partition-based training and aggregation (§ 3.4). We explain the set up in § 3.1 and elaborate the details of Steps 1 to 8.

**Step 4:** The UCD performs backward pass through the classifier $C$ on data $D_C$, while the encoder $E$ is frozen. Then, the UCD updates the parameters of $C$ via computed gradients. Notice that, in practice, the gradient computation done in Step 2 can be reused for this update.

**Step 5:** UCDs share their updated classifier $C$ to the server, and the server performs aggregation on all $C$ from participating UCDs to obtain one aggregated classifier.

**Step 6:** The server sends aggregated classifier back to the UCDs and APs.

**Step 7:** With the updated classifier, the AP trains the encoder $E$ together with the classifier $C$ atop $E$ on data $D_M$. That is, $D_M$ is fed for a forward pass and then a backward pass throughout to update $M$.

**Step 8:** The updated $M$ (including $E$ and $C$) is shared from the AP to the server. After collecting updated $M$ from all participating APs, the server performs model aggregation to obtain the aggregated encoder $E$ and classifier $C$ and distribute them back to UCDs and APs as in Step 6. See § 3.4 for more details of FL training procedures. Steps from 2 to 8 will be repeated for several FL rounds to update $C$ on UCDs and $E$ on APs.

### 3.2 Model Initialization

To partition a DNN into an encoder $E$ and a classifier $C$, one option is to use pre-trained benchmark DNNs; considering numerous well-trained models, e.g., in Model Zoo [25]. Usually, DNN architectures from a candidate space of pre-trained models have already been well developed, with efficient quantization and compression capabilities, to be deployed on UCDs for inference [3, 16, 29, 32, 40, 60]. We aim to utilize the
full capability of UCDs for on-device training of the classifier \( C \). Thus, we need to design the classifier such that it can run on the limited memory and computation power available on these devices.

To achieve these, we can iteratively look into candidate architectures that fit in the limited memory available on these UCDs. A robust mechanism in this direction is to obtain a history of memory usage of a particular device which reflects the typical memory availability on the UCD. Formally speaking, the iterative process can be viewed as an optimization problem described as

\[
C = \min\{m_1, m_2, \ldots, m_k | \mathbb{P}(m_k) \times B < M\},
\]

where \( \mathbb{P}(m_k) \) provides the number of parameters of the candidate architecture \( m_k \), \( B \) is the number of bytes used to store the parameters and intermediate results in computations, and \( M \) is the memory available in the device for FL participation. The parameter \( M \) can be measured and logged as empirical observations of the UCD memory usage, to be sent to the server as we discussed in Step 1 in Figure 4.

In this paper, we limit our classifier architectures with fully connected layers; thus, we find a solution by either reducing the number of neurons or the layers to find a potential architecture for \( C \) that can fit into the requirements for \( M \) in Equation (1). In the same vein, we tackle the problem of low computing resources by choosing the smallest architecture for \( C \) and then deploying it on the UCDs.

### 3.3 Data Selection

We determine the importance of each data sample, along the forward pass, to decide whether to further perform (the more costly) backward pass on this sample or not. Such data selection satisfies design requirements by improving both storage efficiency and training efficiency. We use a combination of (i) the loss value and (ii) the norm of last-layer gradients to measure the sample’s importance.

#### 3.3.1 Loss-based Selection

All available data \( D \) at round \( r \) are fed into the model \( M \) to compute their loss values \( \ell_r(f_M(X), y) \). We consider lower loss values as an indication of being less important, and higher loss values show the importance of the data [22].

To this end, at round \( r \) for the current \( \ell_r \), we drive a cumulative distribution function (CDF\(_r\)). Then at round \( r + 1 \), the probability of discarding data, \( P_N \), and the probability of feeding data to train the complete model \( M \) on AP, \( P_M \), are defined as:

\[
\begin{align*}
P_N(\ell_{r+1}) &= 1 - \left[ \text{CDF}_r(\ell_{r+1}) \right]^\alpha \\
P_M(\ell_{r+1}) &= \left[ \text{CDF}_r(\ell_{r+1}) \right]^\beta
\end{align*}
\]

where \( \alpha \) and \( \beta \) parameters determine the level of selectivity. At round \( r + 1 \), for a sample with loss value \( \ell_{r+1} \), the sample is selected into \( D_N \) with probability of \( P_N(\ell_{r+1}) \). This implies that the lower the loss value for a sample, the greater the probability it will be discarded. Similarly, \( P_M \) is used to select samples with the highest losses as \( D_M \); which means that the higher the loss value for a sample, the greater the probability it will be chosen for training the entire model.
Finally, the rest of the samples that are not selected for $D_N$ or $D_M$ are added to $D_C$ to locally train $C$ on the UCD. We use a fixed-size queue for the CDF to dynamically keep track of loss values so that the $P_N$ and $P_M$ for the current sample can be efficiently computed using its loss and to save computational resources similar to prior work [27].

### 3.3.2 Gradient-based Selection

For samples that are not selected through $P_N$ and $P_M$, and to avoid consuming extra resources of UCDs, we can further compute the gradients of classifier’s last layer. The norm of these gradients gives us a useful hint about the sample’s importance while requiring much less computations than computing all layers’ gradients [24]. The last layers’ norm at round $r$ is produced during training of the classifier by $g_r = \frac{\partial L}{\partial W_r}$, where $W_r$ is the weights of the classifier $C$’s last layer at round $r$. Following the same idea, we derive a CDF for gradient norm values and build a queue to keep track of the computed norm of gradient values. Samples with larger gradients have larger impacts on the model’s weights; thus, we also keep these samples for training the entire model $M$ as $E$ might learn new “features” from them. Thus, the probability of adding a sample to $D_M$ at round $r + 1$ is defined as:

$$P^+_M(\|g_{r+1}\|) = [CDF^\gamma(\|g_{r+1}\|)]^\gamma$$

where $\gamma$ is used for customizing the selection rate. Samples with high norm are selected with $P^+_M(\|g_{r+1}\|)$ and then added to data $D_M$. Our dynamic strategy, in combining loss and last-layer gradient norm, enables Centaur to achieve a better trade-off between computation cost and selection performance on UCDs. Moreover, one may substitute this module with a different data selection technique that could also be dependent upon the specific use case.

### 3.4 Partition-based Training and Aggregation

The training process is conducted on both UCDs and APs. While UCDs only train the classifier $C$ on data $D_C$, APs train the complete model $M$ on data $D_M$. Such procedure allows the model to be (partially) updated when UCDs are offline, in addition to the full updates when UCDs have a connection to the internet. This better utilizes UCDs’ spatiotemporal richness and consequently improves the training efficiency in our design requirements.

Algorithm 1 shows the process of partition-based training and aggregation in Centaur. We set the algorithm to run $\frac{R}{2}$ rounds, considering that UCD training and AP training proceed iteratively in one round. All training is conducted on the client’s devices either on UCDs or APs. Then with all updated weights, the server performs Federated Averaging (FedAvg) [44] to obtain one aggregated global model. The new global model needs to be distributed to all clients UCDs and APs such that (i) the training on AP has an updated classifier, and (ii) the training on UCD in the next round has both an updated encoder and classifier.
Algorithm 1 Multidevice FL with Data Selection and Partition-based Training

1. **Inputs:** (1) \(\mathcal{A}\): the set of \(A\) clients each having two devices, one UCD and one AP, (2) \(D^k\): local dataset for each client \(a \in \{1, \ldots, A\}\), (2) \(f_E\): the encoder part with parameters \(W_E\), (3) \(f_C\): the classifier part with parameters \(W_C\), (4) \(R\): the total number of FL rounds.

2. for \(r \in \{1, \ldots, \frac{R}{2}\}\) rounds do
   3. \(K\) clients \(\leftarrow\) randomly select \(K\) clients from a total of \(A\), using uniform sampling

   — **Training on UCDs** —
   4. for \(k \in \{1, \ldots, K\}\) do
   5. \(X^k, y^k \leftarrow D^k\) (all the samples in the local dataset of client \(k\) UCD)
   6. \(\ell^k = L(f_E \circ f_C(X^k), y^k)\) (forward pass to compute per-sample loss values)
   7. \(D^k_N, D^k_C, D^k_M \leftarrow\) based on \(\ell^k\), perform data selection as detailed in Section 3.3
   8. for all \((X^k_j, y^k_j) \in D^k_C\) do
   9. \(g^k_C = \frac{\partial \ell^k}{\partial W_C}\) (compute \(f_C\)'s gradients on \(D^k_C\) to update \(f_C\))
   10. \(W^k_C = \text{Optimizer}(W^k_C, g^k_C)\) (update \(f_C\)'s parameters using gradients)
   11. \(D^* = \) based on \(g^k_C\), perform data selection as detailed in Section 3.3
   12. \(D^k_M = D^k_M \cup D^*\) (enhance \(D^k_M\) by high-value gradients data points)

   — **Server Side Aggregation on the Classifier Part** —
   13. \(W_C \leftarrow \frac{1}{K} \sum_{k \in K} W^k_C\)
   14. \(W^k_C \leftarrow W_C\) (update all clients with new \(W_C\))

   — **Training on APs** —
   15. for \(k \in \{1, \ldots, K\}\) do
   16. \(X^k, y^k \leftarrow D^k_M\) (all selected samples for training \(f_E\) on client \(k\) AP)
   17. \(\ell^k = L(f_E \circ f_C(X^k), y^k)\) (forward pass to compute per-sample loss values)
   18. \(g^k = \frac{\partial \ell^k}{\partial (W_E, W_C)}\) (compute gradients on \(D^k_M\) to update \(f_E\) and \(f_C\))
   19. \(W^k_E, W^k_C = \text{Optimizer}(W^k_E, W^k_C, g^k)\) (update all parameters using gradients)

   — **Server Side Aggregation on both Encoder and Classifier Parts** —
   20. \(\{W_E, W_C\} \leftarrow \frac{1}{K} \sum_{k \in K} \{W^k_E, W^k_C\}\)
   21. \(\{W^k_E, W^k_C\} \leftarrow \{W_E, W_C\}\) (update all clients with the new aggregated model)

   return \(\{W_E, W_C\}\)

4. **METRICS**

To evaluate the performance of our FL framework, compared to other FL alternatives, we use the following metrics:

1) **Accuracy.** This is the classification accuracy of the trained model on a test set hosted by the central server. For a model obtained at the end of each training round, the test accuracy is computed as

\[
\text{acc} \equiv \frac{1}{T} \sum_{i=1}^{T} \mathbb{1}\left(\arg\max (f_E \circ f_C(X_i)) = y_i\right),
\]
where test set has $T$ pairs of $(X_i, y_i)$ and $\mathbb{1}(C)$ denote the indicator function that outputs 1 if condition $C$ holds.

2) Multiply–Accumulate (MAC). This operation is a common step that computes the product of two numbers and adds that product to an accumulator ($a \leftarrow a + (b \times c)$); a fundamental operation for both any DNN layers during training and inference. We use fvcore [15] library to compute the number of MAC operations. Since fvcore only supports counting MACs in a forward pass, and to count MACs in a backward pass, we use the heuristic that FLOPs (i.e., double MACs) ratio of the backward-forward pass is typically between $1\times$ and $3\times$ and most often is $2\times$ based on models’ specific layer types, according to previous observations [42].

3) Bandwidth. As the model size (i.e., the encoder or the classifier) can be different in each FL round, we use fvcore to count the number of parameters that are communicated in each round. Based on the model size and number of communications among UCDs, APs, and the server, we compute the amount of bandwidth that is consumed. When it is a case, we also count the number of sample points that are uploaded from UCDs to APs.

4) Latency. The latency usually has a linear relation with MAC operations; due to the lack of specialized accelerators [32, 34]. Thus, we estimate the latency of model training based on the processor’s frequency:

$$\text{Latency} = c \cdot \frac{\text{Total MACs}}{\text{Processor Frequency}}.$$ (4)

The ratio between MAC operations and the processor’s instructions, $c$, is typically between 1 and 2 based on specific instruction sets/compilers. For simplicity, we assume that each MAC operation translates to two instructions in an MCU (i.e., $c = 2$). Notice that existing processors (e.g., Intel’s Load Effective Address) completes one MAC in one instruction [59]), and this only scales the experimental results and does not change the final conclusions. Finally, the latency of communication is estimated based on the total amount of data needed to be transmitted, divided by the up-link speed or the down-link speed of devices.

5) Energy. This is the total execution time multiplied by the processor’s consumed power per unit of time. The energy consumption of communication can also be calculated based on the total time of transmitting data multiplied by the transmitters’ power per unit of time. We set the values for UCDs and APs in our simulations as follows:

(A) As computation resources, we set up:

| Device | CPU Frequency (MHz) | Storage Capacity (MB) | Power Capacity (mW/MHz) | Timeout (s) |
|--------|---------------------|-----------------------|-------------------------|-------------|
| UCD    | 100                 | 5                     | 0.05                    | Inf         |
| AP     | 2000                | 4096                  | 1.5                     | Inf         |

(B) For communication resources, we set up:

| Device | Uplink (Mbit/s) | Downlink (Mbit/s) | Communication Energy (W) | Disconnect Probability |
|--------|-----------------|-------------------|--------------------------|------------------------|
| UCD    | 2               | 2                 | 0.0001                   | 0.5                    |
| AP     | 10              | 100               | 10                       | 0                      |
5 EXPERIMENTAL SETUP

Datasets. We use six commonly used datasets. CIFAR10 and CIFAR100 [26] that both include 50K training samples and 10K test samples, with 10 and 100 classes respectively. EMNIST [10] that includes 112K training samples with 47 classes. UCIHAR [2] is a widely used dataset of 30 users performing 6 daily activities; data from the accelerometer and gyroscope sensors were collected by a smartphone worn on the waist. Data from 21 users are used for training and that of the other 9 users for testing purposes. MotionSense [41] also includes accelerometer and gyroscope data from 24 users with a smartphone in the pocket of the trousers who performed 6 activities in 15 trials. We use as test data one trial session for each user and as training data the remaining trial sessions (e.g., one trial of “walking” of each user is used as test data and the other two trials are used as training). PAMAP2 [51] dataset contains data of 13 different physical activities, performed by 9 subjects wearing 3 devices and a heart rate monitor. We use the training-test split provided by PersonalizedFL library [37].

Models. For image classification datasets, we select four DNNs commonly used in mobile/edge-oriented literature: (1) EfficientNet-v2 [54], (2) MobileNet-v3 [19], (3) MNASNet [53], and (4) ShuffleNet [39], with 5.3M, 2.2M, 2.5M, and 1.4M number of parameters respectively. For human-activity recognition datasets, we borrow the ConvNet architecture proposed in [8]. All these models consist of an encoder made of multi-layer CNNs followed by one classifier.

FL Settings. We use Flower [5], a customizable open-source FL framework (Python v3.7, Ray v1.11, and Torch v1.12). We run all simulations on a server with 80 Intel Xeon(R) E5-2698 CPUs, 8 Tesla V100 GPUs (16GB), and 504GB system RAM. For all simulations, we use 100 communication rounds, where at each round, UCDs and APs successively train the model. Unless specified otherwise, we consider 100 clients (each having one AP and one UCD). We randomly sample 10% of the clients for training in each round in our initial experiments and later show that our results hold when the client participation ratio is gradually increased to 100%. Data (D) are partitioned using Latent Dirichlet Allocation (LDA) [4, 7] without resampling (LDA-Alpha=1000). We refer to prior research [7] about details of LDA’s generative process. After such partition, each client owns \( \frac{|D|}{100} \) local samples.

For each client, we randomly select half of the samples and use them for local training of the model. For both APs and UCDs, we consider by default the number of epochs to be 3 with a batch size of 64. The other half of the samples are available with more offline time to simulate the further collection of extra data. APs and UCDs also train more epochs on the extra data while offline. In this setup, we define every unit offline time contributing to one more epoch of training on 20% of these extra local samples. For Equation (2) and (3) in data selection we set \( \alpha, \beta, \gamma \) as 5, 3, 0, respectively. Figure 5 gives the test accuracy when we set different values for the \( \alpha, \beta, \) and \( \gamma \) in the data selection scheme. The ▼ point in the left figure is the UCD training, and the ▲ point in the right figure is the AP training. It is found that with data selection, Centaur can always achieve higher accuracy than both UCD training and AP training.
Baselines. (1) AP Training. In this baseline approach we consider UCDs as data collection apparatuses only and therefore, UCDs upload collected and stored data samples to their connected APs, so that the training only happens on APs. In this case, the complete model (both the encoder and classifier) is updated because APs in general are considered to have sufficient resources. (2) UCD Training. UCDs do not upload the collected data samples. Instead, they conduct training on the data locally. However, as UCDs are typically resource-constrained, only the classifier is considered to be trained on-device. This means that the encoder part is considered to be frozen in this setting.

Mobility Model. In our case, the idea of the mobility model signifies how the mobility patterns of the UCDs (or users) impact the connectivity of the UCD to its AP and eventually to the internet. We define an exclusive-Online Association Matrix $\Omega$, which is a binary matrix representing the user’s exclusive location at a given time instance. Mathematically, $\Omega$ is a binary matrix $\mathcal{T} \times \mathcal{S}$ where $\mathcal{T}$ and $\mathcal{S}$ represent the temporal and spatial granularity of $\Omega$ matrix, respectively. For example, the rows may represent the time zones of the day (early morning, morning, afternoon, evening, and late night), during which the user moves between three locations like home, office, and a public park, represented by the columns of $\Omega$. Furthermore, we ensure that each row’s sum equals unity, which means the user is present exclusively at a unique location in a given temporal instance. Finally, to simulate the connectivity patterns, we define a connectivity matrix $\lambda$, a single row-vector of dimension $\mathcal{S}$, denoting the connectivity probability across different locations. More specifically, for evaluating Centaur during mobility, we generate the global connectivity matrix $\lambda$ and the generate matrix $\Omega$, both chosen uniformly at random, for each user to simulate the mobility scenario.
6 EVALUATION RESULTS

We present the results when running Centaur compared to conventional FL training methods. We also analyze the impact of data/participation heterogeneity and spatiotemporal coverage on training methods.

6.1 Model Accuracy

We use the CNN backbone of the four benchmark models described in §5 as the encoders. We examine three classifiers: small that is only one fully connected (FC) layer of size $z$ (number of classes), medium that has two FC layers of size 64 and $z$, and large that has two FC layers of size 128 and $z$. 

**Centaur outperforms AP training and UCD training for different encoders.** We permute the four encoders and three classifiers and report the test accuracy of Centaur compared to the two other baselines: AP training and UCD training. In Figure 6 we report the highest test accuracy for each encoder across all three classifiers for three datasets. Results show that Centaur outperforms AP training by $0.53\% \sim 40.15\%$ and UCD training by $0.45\% \sim 17.78\%$, depending on the settings. There are several settings in which all FL training methods cannot reach a good accuracy, e.g., NASNet and ShuffleNet on CIFAR100, probably because of their relatively small model sizes compared to data complexity. Among them, Centaur still achieves better performance.

**Centaur outperforms UCD training for different classifiers.** In Figure 7 we present the accuracy of different classifiers for each encoder. In the top plot, the results of training on CIFAR10 and CIFAR100 with an accuracy higher than $30\%$ are presented for better visibility. In the bottom plot, we report the results of UCIHAR, MotionSense, and PAMAP2. The results show that Centaur outperforms both UCD training in

![Accuracy of the best classifier for four different encoders](image)
all classifiers’ sizes. The test accuracy also tends to be similar across small, medium, and large classifiers. In addition, the classifier’s sizes may have less impact on more sophisticated encoders (e.g., EfficientNet and MobileNet), which is also observed in a previous work [46]. This may be because an appropriate encoder already produces high-quality features for unseen data that are easy to learn (e.g., CIFAR10), and in such a case, classifier sizes do not make any difference in test accuracy. Similar patterns are observed in HAR datasets. Notice that for HAR datasets, there is currently a lack of publicly accessible pre-trained encoders to start FL with. As a result, we can observe a more significant performance gap compared to image classification, particularly for more complex datasets, e.g., PAMAP2. This lack of pre-trained encoders can also be the reason that, in HAR datasets, the performance of Centaur and AP are almost the same.
6.2 Efficiency

Cost-Accuracy Trade-off. We use CIFAR10, and fix MobileNet-v3 as the encoder and medium size for the classifier. We compute the test accuracy for a range of MAC, Bandwidth, Latency, and Energy budgets. The average size of each sample of CIFAR10 is 30KB. As shown in Figure 8 (top), on both UCD and AP and for all ranges of accuracy, Centaur achieves lower MAC and Latency than both UCD training and AP training. In addition, Figure 8 (bottom) shows that communication Bandwidth and Latency of Centaur is almost the same as AP training; however Centaur can achieve a higher test accuracy along with more training steps while consuming more resources. Centaur causes much more communication cost than UCD training, because the training only involves the transmission of the classifier, which is much lighter than the encoder and data samples transmitted by Centaur. However, the accuracy of UCD training cannot go further than 83.10%, while Centaur can reach up to 89.90% accuracy. We remark that model training consumes much more energy and causes much more latency than communication.

Correlation among Cost Metrics. To show the connection between cost metrics, we plot MAC, Latency, and Energy in Figure 9. We observe that both Latency and Energy have linear relationships with MAC. Besides, Latency is also linearly correlated with Energy, in such a way that these two entirely overlap when we appropriately scale the y-axes. We remark that Bandwidth also has a similar correlation. Such linear
relationships are because of the assumption we made when computing Latency and Energy; however, the actual cost and the relationship among these metrics might not deviate much in practice.

Based on the above-mentioned correlation, we can add Latency (or Energy) of model training as well as the communication Bandwidth to get the overall workload. The results shown in Figure 1 indicate that the workload still follows a similar pattern as the computation in model training, because the training workload significantly overweight the communication workload.

6.3 Data and Participation Heterogeneity

**Centaur is robust to data heterogeneity.** To create imbalanced non-IID data partitions among clients, we use LDA as defined in §4. Next, we set different values for LDA-Alpha to manipulate the levels of non-IID data partitions. Figure 10 and Figure 11 show the results when LDA-Alpha are changing from 0.001, to 0.1, to 1, and to 1000. The smaller is LDA-Alpha, the less balanced is the dataset. We also annotate some examples of class distribution in Figure 10 demonstrating that LDA-Alpha= 0.001 generates almost 1 class per client, while LDA-Alpha= 1000 generates almost uniform classes per client. Results show that in all cases Centaur can reach higher test accuracy than both AP and UCD training. We remark that due to the retraining of the encoder, both Centaur and AP training can cause more fluctuations than UCD training; this is especially obvious when data distribution tends to be IID (e.g., when LDA-Alpha= 1000 in Figure 10).

**Centaur scales well with higher total and participating clients.** We further scale the FL training problem with the number of total clients from 10 to 1000. 10%, 20%, and 50% of them are selected respectively as participating clients in each round. Figure 12 shows Centaur achieves higher test accuracy compared to both UCD training and AP training. Specifically, with a higher participation rate, all training methods tend to have better performance. However, with a much larger number of total clients (e.g., 1000), the test accuracy is reduced, because the same size of data is partitioned into more portions. Also, with such more fragmented data partitioning, Centaur has much accuracy gain when compared to UCD training.

6.4 Performance of Centaur under Mobility

We evaluate situations when the connection probability $\lambda$ changes based on the mobility model defined in §4. For a more readable visualization, we bucket $\lambda$ in three different ranges of $\lambda \in [0.1,0.4]$, $\lambda \in [0.4,0.7]$, and $\lambda \in [0.7,1.0]$ and report the corresponding Accuracy of Centaur when the UCDs are mobile. Results reported in Figure 13 show that Centaur always has higher efficiency than standard UCD training while incurring a higher latency than direct AP training. Specifically, regarding the much important UCD side, Centaur gains 6.45%, 5.64%, and 4.11% higher accuracy with 51.73%, 60.48%, and 68.35% lower cost both in terms of energy and latency across the different ranges of $\lambda$.

Interestingly, we also observe a rise in accuracy with lower connectivity probability. The primary reason is attributed to the overall design of our experimental setup whereby we make a realistic assumption that when mobile with limited connectivity, these UCDs can gather more data and perform the data selection.
Additionally, with this judiciously selected data, the device trains the classifier for an extended number of epochs thus boosting its accuracy.

7 DISCUSSION AND RELATED WORK

Multitier FL. Compared to the default client-server FL, client-edge-server FL [1, 36] considers a middle layer, e.g., a cellular base station, to orchestrate FL training across clients and a central server. In such a setup, clients in the last tier participate with several devices [9, 13, 62]. This is shown to achieve better trade-offs between accuracy and bandwidth consumption. However, these works assume local training of the whole model on the client devices and do not utilize the capabilities of a resourceful edge device for aiding a constrained device during training, as they are only used for aggregation/communication. This makes it impractical in scenarios involving UCDs.

On-device Training. Most prior works only consider the challenges of on-device inference (i.e., forward pass) [3, 16, 32, 60], or work on sparse training [6, 29]. Sparse training or pruning still requires computational resources (especially memory) that UCDs do not have. Few works propose solutions for full on-device training (i.e., forward and backward pass) [34], but not for FL scenarios as they often make impractical assumptions like the availability of large datasets for network architecture search [32] or unconstrained memory for hosting the full model [13, 62]. Transfer learning can be used on UCDs to fix the encoder architecture, which in turn restricts the extraction of newer features, and only trains the last layer for the personalized training [62]. Regarding complex tasks, training the fully functional model on UCDs is challenging, because the on-device model still needs the capability of learning all features by itself [6, 29], and the computational resources at the upstream devices are unutilized.
Fig. 10. The impacts of imbalanced data partitioning on the test accuracy of AP training, UCD training, and Centaur, reported on MobileNetv3 on CIFAR10 dataset (annotated texts give examples of the classes present on the UCDs).

Fig. 11. Impacts of imbalanced data partitioning on the test accuracy of AP training, UCD training, and Centaur, reported on ConvNet and UCIHAR dataset dataset.
Fig. 12. The impact of participating clients and total clients on test accuracy reported on MobileNetv3 and CIFAR10.

Communication/Power Efficiency. Using a pre-trained encoder without further training might fail to capture discerning features from the newly generated data, causing lower performance for the ultimate classifiers. On the other hand, passing all collected sensor data from UCDs to APs (even with data compression) is not always feasible and it might consume a lot of energy and bandwidth. Importantly, network availability for UCDs can be fragile due to the low-energy wireless communication protocols they use and the mobility of these UCDs. As UCDs are becoming stand-alone, it is expected that they can operate as independent devices without internet or permanent connection to an upstream device. Centaur takes a major step in unlocking more functionalities of such development. Compression techniques [33] or topology-aware hierarchical aggregation [38] can reduce Latency by taking advantage of the locality in networks and continuously balancing workloads across clients. However, compression and hierarchical aggregation alone do not allow for seamlessly incorporating UCDs in FL. Our proposed method comprises different methods to offer customizable trade-offs between test accuracy and communication cost in data- and topology-agnostic manners, while allowing additional use of the recent compression and hierarchical aggregation techniques for further efficiency.

Training on UCDs. By default, FL algorithms are not specifically designed for a multi-device environment [21, 28]. Training DNNs on tiny MCU devices is extremely challenging due to limited memory, computation, and power resources. There are recent efforts to enable running ML models on a microcontroller with highly limited memories, such as TensorFlow Lite Micro [11] or Tiny Training Engine [34]. Such developments, usually based on a combination of hardware and software novelties, can be easily integrated into Centaur to provide further efficiency. As we observe, a better training framework can not only enable deploying larger classifiers on UCDs, but can also allow us to calculate more gradients for
better data selection. While it is difficult to train large DNNs on UCDs, these devices are typically parts of a personal network, and therefore, they can privately communicate/coordinate with other surrounding devices. Our work considers such a hierarchy for unlocking more utilities while providing better efficiency.

8 LIMITATIONS AND FUTURE WORK

First, we accommodate FedAvg [44] in our current implementation, but Centaur is compatible with other FL algorithms, e.g., FedProx [31], FedAdam [50], or DM-PFL [64]. On the other hand, layer-wise FL methods, such as FedMA [57], may have interesting interactions as they also conduct model training in a partitioned fashion.

Second, more algorithmic changes can be done to existing AP training, but we emphasize that whatever we do for AP training can only be useful for applications where all data is available at AP. In many emerging applications, UCDs are the main devices that collect and process data in the first place, and we do not want UCDs to send all the data to AP and put the burden of data selection on the AP's shoulders.

Third, we do not perform model selection or architecture search, and our model partitioning simply follows the encoder-classifier style of benchmark networks, but one can enhance the model initialization and partitioning with state-of-the-art advancements in neural architecture search [20, 30].

Fourth, our goal is to present a proof of concept for a multi-device FL that encompasses UCDs and APs. We present a real testbed and perform actual measurements, but in the rest of our experiments, we utilize
GPU servers for running end-to-end FL scenarios. The constraints of UCDs are simulated according to the size and type of the model they can train. As a topic for future investigation, and to achieve more precise calculations of compute and storage resources, the end-to-end process of Centaur could be implemented on real-world UCDs and APs.

Fifth, we considered a one-to-one relationship between UCDs and APs, but one can expand Centaur into a many-to-one setting. This, however, needs tackling the challenge of aggregating diverse classifiers trained by various sensing devices.

Sixth, we evaluated Centaur on supervised FL tasks, but it is possible to suggest a supplementary approach for implementing semi-supervised or unsupervised tasks. Loss-based data selection has been studied very well in the ML community and shows promising advantages. Finally, dealing with outliers and distribution shifts are important challenges but we believe these are topics outside the scope of this paper.

9 CONCLUSIONS

To include constrained devices in an FL task, we offer an efficient training approach that utilizes data selection to enhance learning while minimizing computational costs. Additionally, we employ partition-based training and aggregation, wherein constrained devices train a specific part of the model, aided by more capable devices that train the complete model. Our assessments validate the benefits of our proposed FL framework, Centaur, in enhancing model accuracy while simultaneously reducing system overhead. Furthermore, we propose several promising directions for future studies in the area of efficient FL on constrained sensory devices.

REFERENCES

[1] Mehdi Salehi Heydar Abad, Emre Ozfatura, Deniz Gunduz, and Ozgur Ercetin. 2020. Hierarchical federated learning across heterogeneous cellular networks. In ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 8866–8870.

[2] D. Anguita, Alessandro Ghio, L. Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. 2013. A Public Domain Dataset for Human Activity Recognition using Smartphones. In The European Symposium on Artificial Neural Networks.

[3] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. 2021. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. Proceedings of Machine Learning and Systems (MLSys) 3 (2021), 517–532.

[4] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, et al. 2022. Flower: A friendly federated learning framework. (2022).

[5] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D Lane. 2020. Flower: A Friendly Federated Learning Research Framework. arXiv preprint arXiv:2007.14390 (2020).

[6] Sameer Bibikar, Haris Vikalo, Zhangyang Wang, and Xiaohan Chen. 2022. Federated dynamic sparse training: Computing less, communicating less, yet learning better. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36. 6080–6088.
Enhancing Efficiency in Multidevice Federated Learning through Data Selection

[7] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.

[8] Youngjae Chang, Akhil Mathur, Anton Isopoussu, Junehwa Song, and Fahim Kawsar. 2020. A systematic study of unsupervised domain adaptation for robust human-activity recognition. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 1 (2020), 1–30.

[9] Hyunsung Cho, Akhil Mathur, and Fahim Kawsar. 2022. Flame: Federated learning across multi-device environments. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 3 (2022), 1–29.

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

[11] Robert David, Jared Duke, Adwait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. 2021. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems (MLSys)* 3 (2021), 800–811.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

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

[14] Fotios Drakopoulos, Deepak Baby, and Sarah Verhulst. 2019. Real-time audio processing on a Raspberry Pi using deep neural networks. In *23rd International Congress on Acoustics (ICA 2019)*. Deutsche Gesellschaft für Akustik, 2827–2834.

[15] FAIR. 2019. fvcore. https://github.com/facebookresearch/fvcore

[16] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. 2019. Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems 32* (2019).

[17] Otkrist Gupta and Ramesh Raskar. 2018. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* 116 (2018), 1–8.

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

[19] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*. 1314–1324.

[20] Kai Huang, Boyuan Yang, and Wei Gao. 2023. ElasticTrainer: Speeding Up On-Device Training with Runtime Elastic Tensor Selection. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. 56–69.

[21] Bashima Islam and Shahriar Nirjon. 2019. Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems. *arXiv preprint arXiv:1905.03854* (2019).

[22] Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminsky, Michael Kozuch, Zachary C Lipton, et al. 2019. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762* (2019).

[23] Peter Kairouz, H. Brendan McMahan, Brendan Avent, and et al. 2021. Advances and Open Problems in Federated Learning. *Foundations and Trends® in Machine Learning* 14, 1–2 (2021), 1–210.

[24] Angelos Katharopoulos and François Fleuret. 2018. Not all samples are created equal: Deep learning with importance sampling. *arXiv preprint arXiv:1905.03854* (2019).

[25] Peter Kairouz, H. Brendan McMahan, Brendan Avent, and et al. 2021. Advances and Open Problems in Federated Learning. *Foundations and Trends® in Machine Learning* 14, 1–2 (2021), 1–210.

[26] Jing Yu Koh. 2019. ModelZoo. https://modelzoo.co/

[27] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
[28] Seulki Lee, Bashima Islam, Yubo Luo, and Shahriar Nirjon. 2019. Intermittent learning: On-device machine learning on intermittently powered system. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 4 (2019), 1–30.

[29] Ang Li, Jingwei Sun, Pengcheng Li, Yu Pu, Hai Li, and Yiran Chen. 2021. Hermes: an efficient federated learning framework for heterogeneous mobile clients. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 420–437.

[30] Ang Li, Jingwei Sun, Binhui Wang, Lin Duan, Sicheng Li, and Hai Li. 2021. LotteryFL: empower edge intelligence with personalized and communication-efficient federated learning. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 68–79.

[31] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *Proc. Conf. on Machine Learning and Systems (MLSys)*.

[32] Edgar Liberis, Łukasz Dudziak, and Nicholas D Lane. 2021. \(\mu\)NAS: Constrained Neural Architecture Search for Microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems*. 70–79.

[33] Hyeontaek Lim, David G Andersen, and Michael Kaminsky. 2019. 3LC: Lightweight and effective traffic compression for distributed machine learning. *Proceedings of Machine Learning and Systems (MLSys)* 1 (2019), 53–64.

[34] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. 2022. On-Device Training Under 256KB Memory. *arXiv preprint arXiv:2206.15472* (2022).

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

[36] Lumin Liu, Jun Zhang, SH Song, and Khaled B Letaief. 2020. Client-edge-cloud hierarchical federated learning. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[37] Wang Lu and Jindong Wang. 2023. PersonalizedFL: Personalized Federated Learning Toolkit. https://github.com/microsoft/PersonalizedFL.

[38] Liang Luo, Peter West, Jacob Nelson, Arvind Krishnamurthy, and Luis Ceze. 2020. Plink: Discovering and Exploiting Locality for Accelerated Distributed Training on the public Cloud. In *Proceedings of Machine Learning and Systems*, Vol. 2. 82–97.

[39] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*. 116–131.

[40] Mohammad Malekzadeh, Richard Clegg, Andrea Cavallaro, and Hamed Haddadi. 2021. Dana: Dimension-adaptive neural architecture for multivariate sensor data. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 3 (2021), 1–27.

[41] Mohammad Malekzadeh, Richard G Clegg, Andrea Cavallaro, and Hamed Haddadi. 2019. Mobile sensor data anonymization. In *Proceedings of the international conference on internet of things design and implementation*. 49–58.

[42] Jsevillamol Marius Hobbhahn. 2021. What’s the backward-forward FLOP ratio for Neural Networks? https://www.lesswrong.com/posts/fnjKpBoWJXcSDwZhK/what-s-the-backward-forward-flop-ratio-for-neural-networks.

[43] Akhil Mathur, Daniel J Beutel, Pedro Porto Buarque de Gusmao, Javier Fernandez-Marques, Taner Topal, Xinchi Qiu, Titouan Parcollet, Yan Gao, and Nicholas D Lane. 2021. On-device federated learning with flower. *arXiv preprint arXiv:2104.03042* (2021).

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

[45] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proc. Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*.

[46] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolasourtelis. 2021. FFPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 94–108.
[47] Fan Mo and Jia Zhou. 2021. Adapting smartwatch interfaces to hand gestures during movements: offset models and the C-shaped pattern of tapping. *Journal of Ambient Intelligence and Humanized Computing* 12, 7 (2021), 8099–8117.

[48] Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. 2020. Billion-scale federated learning on mobile clients: A submodel design with tunable privacy. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.

[49] Franz Papst, Katharina Schodl, and Olga Saukh. 2021. Exploring co-dependency of iot data quality and model robustness in precision cattle farming. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 433–438.

[50] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H Brendan McMahan. 2020. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295* (2020).

[51] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th international symposium on wearable computers*. IEEE, 108–109.

[52] Suranga Seneviratne, Yiming Hu, Tham Nguyen, Guohao Lan, Sara Khalifa, Kanchana Thilakarathna, Mahbub Hassan, and Aruna Seneviratne. 2017. A survey of wearable devices and challenges. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2573–2620.

[53] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2820–2828.

[54] Mingxing Tan and Quoc Le. 2021. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*. PMLR, 10096–10106.

[55] Chandra Thapa, Pathum Chamikara Mahawaga Arachchi, Seyit Camtepe, and Lichao Sun. 2022. SplitFed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 8485–8493.

[56] Delia Velasco-Montero, Jorge Fernández-Berni, Ricardo Carmona-Galán, and Ángel Rodríguez-Vázquez. 2018. Performance analysis of real-time DNN inference on Raspberry Pi. In *Real-Time Image and Video Processing* 2018, Vol. 10670. SPIE, 115–123.

[57] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. 2020. Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440* (2020).

[58] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big data* 3, 1 (2016), 1–40.

[59] WiKiBooks. 2021. x86 Assembly/Data Transfer. https://en.wikibooks.org/wiki/X86_Assembly/Data_Transfer

[60] Kunran Xu, Yishi Li, Huawei Zhang, Rui Lai, and Lin Gu. 2022. EtinyNet: Extremely Tiny Network for TinyML. (2022).

[61] Qian Yang, Jianyi Zhang, Weituo Hao, Gregory P Spell, and Lawrence Carin. 2021. Flop: Federated learning on medical datasets using partial networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3845–3853.

[62] Lan Zhang, Dapeng Wu, and Xiaoyong Yuan. 2021. FedZKT: Zero-Shot Knowledge Transfer towards Resource-Constrained Federated Learning with Heterogeneous On-Device Models. *arXiv preprint arXiv:2109.03775* (2021).

[63] Tuo Zhang, Lei Gao, Chaoyang He, Mi Zhang, Bhaskar Krishnamachari, and A Salman Avestimehr. 2022. Federated learning for the internet of things: Applications, challenges, and opportunities. *IEEE Internet of Things Magazine* 5, 1 (2022), 24–29.

[64] Wenhao Zhang, Zimu Zhou, Yansheng Wang, and Yongxin Tong. 2023. DM-PFL: Hitchhiking Generic Federated Learning for Efficient Shift-Robust Personalization. (2023).