DISTREAL: Distributed Resource-Aware Learning in Heterogeneous Systems

Martin Rapp[1], **Ramin Khalili**[2], **Kilian Pfeiffer**[1], **Jörg Henkel**[1]

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany
[2] Huawei Research Center, Munich, Germany
martin.rapp@kit.edu, ramin.khalili@huawei.com, kilian.pfeiffer@kit.edu, henkel@kit.edu

Abstract

We study the problem of distributed training of neural networks (NNs) on devices with heterogeneous, limited, and time-varying availability of computational resources. We present an adaptive, resource-aware, on-device learning mechanism, *DISTREAL*, which is able to fully and efficiently utilize the available resources on devices in a distributed manner, increasing the convergence speed. This is achieved with a dropout mechanism that dynamically adjusts the computational complexity of training an NN by randomly dropping filters of convolutional layers of the model. Our main contribution is the introduction of a design space exploration (DSE) technique, which finds Pareto-optimal per-layer dropout vectors with respect to resource requirements and convergence speed of the training. Applying this technique, each device is able to dynamically select the dropout vector that fits its available resource without requiring any assistance from the server. We implement our solution in a federated learning (FL) system, where the availability of computational resources varies both between devices and over time, and show through extensive evaluation that we are able to significantly increase the convergence speed over the state of the art without compromising on the final accuracy.

Introduction

Deep learning has achieved impressive results in a number of diverse domains, such as image classification (Howard et al. 2017; Tan and Le 2019), board and video games (Silver et al. 2016; Mnih et al. 2016), and is widely applied to distributed systems, such as mobile and sensor networks (Zhang, Patras, and Haddadi 2019), as we consider in this paper. Centralized deep learning techniques, where the training is performed in a single location (e.g., a data center), is often costly, as data would need to be collected and sent all over the network to that centralized entity (Shi et al. 2020), and might not be feasible/authorized if the training uses users' private data. FL (McMahan et al. 2017) emerged as an alternative to such techniques, performing distributed learning on each device with the locally available data.

FL has proven effective in large-scale systems (Bonawitz et al. 2019; Liu, Wang, and Liu 2019; Chen et al. 2020b). However, training of a deep NN model is resource-hungry in terms of computation, energy, time, etc. (You et al. 2018), and it is rather unrealistic to assume that all devices in an FL system can perform all types of training computations all the time, especially if the training is distributed on edge devices, e.g., as suggested for 6G systems.[1] This is as the computational capabilities of devices participating in an FL system may be heterogeneous, e.g., different hardware, different generations (Bonawitz et al. 2019). *More importantly, the resources available on a device for training could change over time*. This could for instance be due to shared resource contention (Dhar et al. 2019), where CPU time, cache memory, energy, etc. are shared between the learning and parallel tasks. We illustrate this with the following two examples.

1) Edge computing has been employed in ML-based real-time video analytics, where each edge device processes images from several camera modules (Ananthanarayanan et al. 2017). Currently, edge devices mostly perform inference, but there is a clear trend towards additionally performing distributed learning via FL (Zhou et al. 2019). The learning task shares computational resources with the inference tasks. The inference workload depends on the activity in the video images and changes over time, as processing is skipped for subsequent similar images to save resources (Ananthanarayanan et al. 2017). These changes happen fast, i.e., in the order of seconds (Zhang et al. 2017), while FL round times may be minutes (Bonawitz et al. 2019). 2) Google *GBoard* (Yang et al. 2018) trains a next-word-prediction model using FL on end users' mobile phones. To avoid slowing down user applications, and thereby degrading the user experience, training is performed only when the device is charging and idle, and aborted when these conditions change. This introduces a bias towards certain devices and users, degrading the model accuracy (Yang et al. 2018). This can be resolved by allowing training also when the device is in use, but only using free resources. Smartphone workloads change within seconds (Tu et al. 2014), which is faster than the *GBoard* round time of several minutes. In both examples, the learning task is subject to fast-changing resource availability.

While several works study the problem of heterogeneity across devices (Li et al. 2020a; Imteaj et al. 2020), time-

[1] EU's Horizon Europe (European Commission 2021a,b) calls for proposals, as well as 5GIA (5G Infrastructure Association) and SRIA (Strategic Research and Innovation Agenda) reports (Bernardos and Uusitalo 2021; NetWorld 2020) detail such a vision.

varying resource availability has so far been neglected. In this paper, we propose a distributed, resource-aware, adaptive, on-device learning technique, *DISTREAL*, which enables us to fully exploit and efficiently utilize available resources on devices for the training, dealing with all these types of heterogeneity. Our objective is to maximize the accuracy that is reached after limited training time on devices, i.e., convergence speed. To fulfill this goal, we should make sure that C1) The available resources on a device are *fully* exploited. This requires fine-grained adjustability of the training model on a device, and a method to instantly react to changes; and C2) The available, limited, resources on a device are used *efficiently*, to maximize the accuracy improvement and hence the overall convergence speed. Specifically, we provide the following novel contributions:

- We introduce and formulate the problem of heterogeneous time-varying resource availability in FL.
- We propose a dropout technique to adjust the computational complexity (resource requirements) of training the model *at any time*. Thereby, each device locally decides the dropout setting which fits its available resources, without requiring any assistance from the server, addressing C1. This is different from the state-of-the-art techniques, such as (Caldas et al. 2018; Horváth et al. 2021; Xu et al. 2019; Diao, Ding, and Tarokh 2021), where the server is responsible for regulating resource requirements of training for each device at the beginning of each training round, which may take several minutes.
- We show that using different per-layer dropout rates achieves a much better trade-off between the resource requirements and the convergence speed, compared to using the same rate at all layers as the state of the art (Caldas et al. 2018; Diao, Ding, and Tarokh 2021), addressing C2. We present a DSE technique to automatically find the Pareto-optimal dropout vectors at design time.

We implement our solution *DISTREAL* in an FL system, in which the availability of computational resources varies both between devices and over time. We show through extensive evaluation that *DISTREAL* significantly increases the convergence speed over the state of the art, and is robust to the rapid changes in resource availability at devices, without compromising on the final accuracy.

System Model and Problem Definition

System Model We target a distributed system, which comprises one *server* and N distributed *devices* that act as clients. Each device i holds its own local training data X_i. The system uses FL for decentralized training of an NN model from the distributed data. We target a synchronous coordination scheme, which divides the training into many *rounds*. At the beginning of a round, the server selects n devices to participate in the training. Each selected device downloads the recent model from the server, trains it with its local data, and sends weight updates back to the server. The server combines all received weight updates to a single update by weighted averaging. Updates from devices that take too long to perform the training (stragglers) are discarded.

Device Resource Model The devices are subject to time-varying limited computational resource availability for training. To which degree the availability of a certain resource affects the training time of an NN depends on the NN and hyperparameters, but also on the deep learning library implementation and the underlying hardware (Chen et al. 2020c). We abstract from such specifics of the hardware and software implementation, and from the constrained physical resource to keep this work applicable to many systems by representing the resource availability in the number of multiply-accumulate operations (MACs) that a device can calculate per time given its specifications and available resources. MAC operations are the fundamental building block of NNs (e.g., fully-connected and convolutional layers) and account for the great majority of operations (Krizhevsky, Sutskever, and Hinton 2012). In the appendix, we also provide experimental evidence for the suitability of MACs/s as an abstract metric. Resource availability varies between devices and over time. Therefore, these resource availabilities $r_i(t)$ depends on the device i, and the current time t. Resources may change at any time, i.e., also within an FL round. Resources are not required to be known ahead of time.

Objective Our objective is to maximize the convergence speed of training, i.e., the reached accuracy after a number of rounds, under heterogeneous (between devices and over time) resource availability.

Related Work

Many works on resource-aware machine learning focus on resource-aware *inference* (Tann et al. 2016; Yu et al. 2018; Amir and Givargis 2018; Li et al. 2021). These techniques allow adapting the inference to dynamically changing availability of resources at run-time but are not applicable to training. Resource-aware *training* is recently getting increasing attention, mostly in the context of FL. Most attention has so far been paid to limited *communication* resources, leading to solutions, such as compression, quantization, and sketching (Shi et al. 2020; Thakker et al. 2019) Importantly, these works do not reduce the *computational* resources for training, as they are applied after local training has finished. These works are complementary/orthogonal to our work and can be adopted to our solution (see the section on the run-time technique). Techniques on computation-resource-aware training can be categorized into two classes: techniques that always train the full NN on each device but with fewer data/relaxed timing and techniques that train subsets of the NN.

Train Full NNs FedProx (Li et al. 2020b) allows devices participating in an FL system to deliver partial results to the server by dropping training examples that could not be processed with the available resources. Our previous work (Rapp, Khalili, and Henkel 2020) studied multi-head networks where each device uses the head that fits its available resources. Devices only synchronize the weights of the first shared layers. However, this technique has low adaptability as only a few resource levels can be supported. Asynchronous variants of FL have been proposed that allow devices to finish training at any time (Chen et al. 2020a; Xie,

Koyejo, and Gupta 2020). However, asynchronous synchronization may reduce the convergence stability (McMahan et al. 2017; Xu et al. 2019). Techniques based on Federated Distillation (Li and Wang 2019; Chang et al. 2019; Lin et al. 2020) synchronize knowledge between devices by exchanging labels on a public dataset instead of exchanging NN weights. Therefore, each device has the design flexibility to use an NN model according to its constraints. However, Federated Distillation cannot cope with time-varying resources.

Train NN Subsets Several techniques perform training only on a dynamic subset of the NN, to be able to fit the resource requirements of training to the resource availability on each device. FjORD (Horváth et al. 2021), Yu and Li (2021), and HeteroFL (Diao, Ding, and Tarokh 2021) select subsets of the NN for each device at the beginning of each round. They select the subsets in a hierarchical way, where smaller subsets are fully contained in larger subsets. HeteroFL introduces a shrinkage ratio s that defines the ratio of removed hidden channels to reduce the resource requirements of the NN. The same parameter s is applied repeatedly to all layers to obtain several subsets with decreasing resource requirements. Using hierarchical subsets restricts the granularity of resource requirements, as increasing the number of supported subsets reduces the achievable accuracy (Tann et al. 2016). This limitation can be avoided by selecting subsets randomly, i.e., use different subsets in every round. ELFISH (Xu et al. 2019) randomly removes neurons before training on slow devices at the beginning of a round. Graham, Reizenstein, and Robinson (2015) study the suitability of dropout (Srivastava et al. 2014) to reduce resource requirements. They find that computations can only be saved if dropout is done in a structured way, i.e., the same neurons are dropped for all samples of a mini-batch. Federated Dropout (Caldas et al. 2018) has been originally proposed to reduce the communication and computation overhead of FL. They perform dropout at the server and train a repacked smaller network on the devices. The dropout masks are changed randomly in each round, which results in all parts of the NN being trained eventually. However, they use the same dropout rate for all devices and a single dropout rate for all layers. All these works select the trained subset at the server, which may reduce the communication volume, but importantly does not allow to adapt to changing resource availability on the devices within a round.

Resource-Aware Training of NNs

Our technique comprises two parts. At run time (online), we dynamically drop parts of the NN using an adapted version of dropout (Srivastava et al. 2014). The Pareto-optimal vectors of dropout rates w.r.t. convergence speed and resource requirements are obtained at design time (offline) using a DSE. Before going into the details of our contribution, we introduce dropout as the basis of our technique.

Dropout to Reduce Computations In Training Dropout was originally designed as a regularization method to mitigate overfitting (Srivastava et al. 2014). It randomly drops individual neurons during training with a certain probability (dropout rate). This results in an irregular fine-grained pattern of dropped neurons. All major deep learning libraries perform dropout by calculating the output of all neurons and multiplying the dropped ones with 0 (Abadi et al. 2015; Paszke et al. 2019). This wastes computational resources; it would be more efficient to not calculate values that are going to be dropped. However, convolutional and fully-connected layers are implemented as matrix-vector or matrix-matrix operations that are heavily optimized with the help of vectorization (Abadi et al. 2015; Paszke et al. 2019). Skipping the calculation of individual values results in sparse matrix operations, which breaks vectorization, increasing the required resources instead of decreasing them (Song et al. 2019).

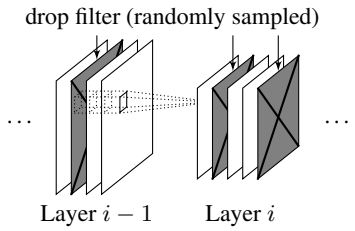


Figure 1: Filter-based structured dropout in a convolutional layer maintains regularity in the calculations while significantly reducing the required computations.

To reduce the number of computations, the dropout pattern needs to show some regularity that still allows using vectorization of dense matrix operations. This can be achieved by dropping contiguous parts of the computation (Graham, Reizenstein, and Robinson 2015). Modern NNs consist of many different layer types such as convolutional, pooling, fully-connected, activation, or normalization layers. Many of these layers are computationally lightweight (e.g., pooling), while some contain the majority of computations (convolutional and fully-connected layers). In state-of-the-art convolutional NNs, the convolutional part requires orders of magnitude more MACs than the fully-connected part. (See the appendix for an experimental analysis.) We, therefore, argue that a technique to save computations needs to target convolutional layers. Fig. 1 depicts filter-based structured dropout in a convolutional layer, as we apply in this paper: instead of dropping individual pixels in the output, whole filters are dropped stochastically. This approach reduces the number of computations while allowing to keep existing vectorization methods.

Fig. 2 depicts how the numbers MACs of the forward pass evolve when we apply different vectors of per-layer dropouts for DenseNet-40 (details in the experimental evaluation). We apply the DSE technique introduced in this paper to determine these vectors and show in the x-axis the resulting ratio of dropped filters[2]. We observe that the number of MACs decreases almost quadratically with this ratio. We also report the training time of a single mini-batch on a Raspberry Pi 4, which serves as an example for an IoT de-

[2]Multiple vectors may results in the same ratio of dropped filters, while providing different convergence / resource requirement trade-offs, explaining why for some ratios we have multiple MACs.

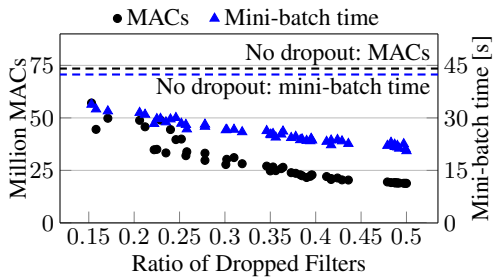


Figure 2: The number of MACs and mini-batch training time decrease quadratically with the ratio of dropped filters.

vice, using an implementation of this structured dropout in PyTorch (Paszke et al. 2019), which is publicly available[3]. A training step (forward pass, backpropagation, and weight updates) requires about $\sim 2\times$ more MACs than the forward pass alone (Amodei et al. 2018). The mini-batch training time shows a similar trend as the number of MACs but with an offset. This is because our implementation does not modify the backend of PyTorch to be aware of this dropout, which results in copy operations of weight tensors to repack them. As a consequence, the measured benefits are smaller than what is theoretically achievable. Changing the backend would get the benefits closer to the optimum but is not easily doable because of closed sources (e.g., of CUDA). In summary, this experiment shows that structured dropout significantly reduces the required computational resources.

The dropout rates also change the convergence speed. A higher dropout rate in a layer means that in each training update, a smaller fraction of the layer's weights is updated, thereby slowing down the training. As a consequence, the dropout rate determines a trade-off between the resource requirements and the convergence speed. The dropout rate should always be selected as low as the available resources allow. To cope with changing resource availability, we propose to *dynamically change the dropout rate at run time*. Inference uses always the whole model.

Design-Time DSE: Find Pareto-Optimal Dropout Vectors The resource requirements (MACs) and convergence speed both depend on the dropout rates of *each layer*. Prior works restrict themselves to choosing a common dropout rate for all layers (Caldas et al. 2018). Relaxing this restriction opens up a larger design space, where each dropout rate of each layer is adjusted towards a better trade-off between resource requirements and training convergence. However, this design space could be too large to be explored manually. For example, DenseNet-100 has 99 convolutional layers that each need to be assigned a dropout rate. Some works apply simple parametric functions of the depth to similar problems (Huang et al. 2016). However, this only works in case of a homogeneous NN structure, where properties of layers (e.g., MACs) change monotonically. For instance, DenseNet layers alternate between computationally lightweight and complex, rendering a simple parametric function sub-optimal. This section describes the required automated DSE technique to efficiently explore such a large space. The DSE is executed only once at design time (offline).

Specifically, the design space contains all combinations of dropout values per layer. We select dropout values from the continuous range $[0, 0.5]$ because higher values reduce the final achievable accuracy, as we observe in our experiments, as well as indicated in previous studies (Srivastava et al. 2014). For an NN with k convolutional layers, the design space is $[0, 0.5]^k$. We have two objectives, the resource requirements and the convergence speed.

Resource Requirements: As discussed in the device resource model, the number of MACs is an implementation-independent representation of the resource requirements. Dropout is a probabilistic process, i.e., the number of MACs varies between different update steps. The resource requirements with a certain dropout vector is represented by the expectation value of the number of MACs of the forward pass. This number can be analytically computed depending on the layer topology, the dropout rate of this layer and preceding layers. The appendix lists equations for different layer types.

Convergence Speed: The convergence speed with a certain dropout vector is measured by observing the accuracy change when training. Exploring the search space takes too long if a full training with every candidate dropout vector is performed. Instead, we assess the accuracy change after a short training, similar to learning curve extrapolation in neural architecture search (Baker et al. 2017). We train for 64 mini-batches with batch size 64, which allows us to explore many candidate dropout vectors in a reasonable time. This corresponds to the amount of data collected by very few devices. To reduce the impact of random initialization, the NN is not trained from scratch but from a snapshot after partially training it on a distorted version of the dataset. For instance, we reduced the brightness, contrast, and saturation to 0.5 of the original value for CIFAR-10/100 datasets. The DSE, therefore, does not require access to the devices' data, but only access to a small amount of similar (or even synthetic) data. To further reduce the impact of random variations, we repeat this with three different random seeds. The convergence speed is represented by the average accuracy improvement.

Fig. 3 shows our DSE flow. The problem of finding Pareto-optimal dropout vectors is a multi-objective optimization. This is a well-studied class of problems with many established algorithms. Evolutionary algorithms have successfully been employed for neural architecture search (Elsken, Metzen, and Hutter 2019), which is related to the problem studied in this section. Note that we are not searching for an architecture, but tune parameters of a given architecture. The output of the DSE is the Pareto-front of dropout vectors. To have a large variety of options to chose from at run time, but also keep a low number of vectors to be stored, the Pareto-front should be approximately equidistantly represented. We use the NSGA-II (Deb et al. 2002) genetic algorithm from the pygmo2 library (Biscani and Izzo 2020) with a population size of 64. NSGA-II explores the search space by crossover (combining parts of two dropout vectors) and mutation (random changes) of good dropout

[3]https://git.scc.kit.edu/CES/DISTREAL

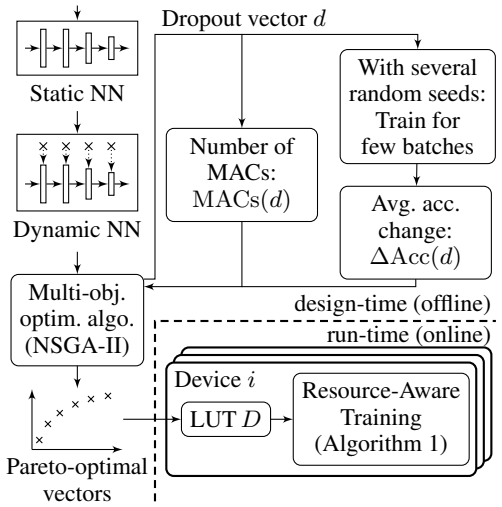


Figure 3: Efficient resource-aware training comprises the DSE to find Pareto-optimal vectors of dropout rates per layer and resource-aware training on each device at run time.

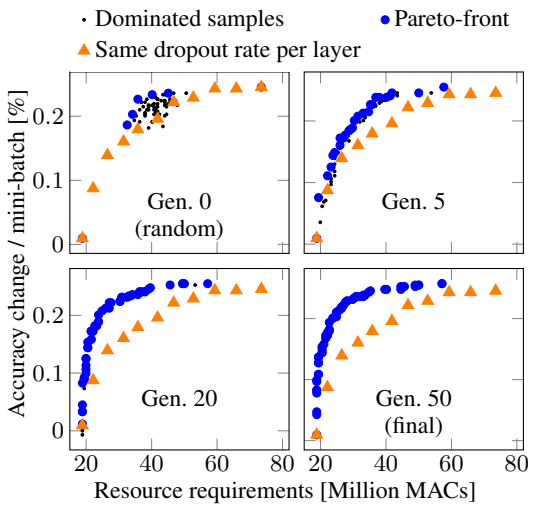


Figure 4: The evolving Pareto-front for DenseNet-40 significantly outperforms setting the same rate for all layers.

vectors w.r.t. the objective function, and is designed to obtain dropout vectors that are equidistantly distributed across the Pareto-front. Thereby, an *individual* is one dropout vector containing the per-layer dropout rates. For our largest studied NN, DenseNet-100, this is 99 float values between 0 and 0.5. A *population* is a set of individuals. We use a population size of 64. A *generation* performs one optimization step on the population with the goal to find the Pareto-front. The optimization minimizes the following two-dimensional *fitness* function $f(d)$ for a dropout vector d, which normalizes the values of the resource requirements $\text{MACs}(d)$ and convergence speed $\Delta\text{Acc}(d)$ to the range $[0, 1]$:

$$f(d) = \begin{pmatrix} \frac{\text{MACs}(d) - \text{MACs}(\{0.5,\ldots,0.5\})}{\text{MACs}(\{0,\ldots,0\}) - \text{MACs}(\{0.5,\ldots,0.5\})} \\ \frac{\Delta\text{Acc}(\{0,\ldots,0\}) - \Delta\text{Acc}(d)}{\Delta\text{Acc}(\{0,\ldots,0\}) - \Delta\text{Acc}(\{0.5,\ldots,0.5\})} \end{pmatrix} \quad (1)$$

Fig. 4 shows the evolving population of dropout vectors for DenseNet-40. The initial population comprises random dropout vectors. We add two samples to the initial population (all dropout values are 0 / 0.5) to accelerate the exploration of the Pareto-front (leverage the crossover operation). After 50 generations of NSGA-II, the Pareto-front has fully evolved and shows a continuous trade-off between resource requirements and convergence speed. Importantly, the Pareto-front found by DSE provides a significantly better trade-off between resource requirements and convergence speed compared to using the same dropout rate for all layers.

Run Time: Resource-Aware Training of NNs After finding the Pareto-optimal dropout vectors, they are stored in a lookup table (LUT) D, along with the corresponding number of MACs. The LUT is small in size (e.g., 25 kB for DenseNet-100 for storing 64 dropout vector of 99 dropout values and the number of MACs, each in 32-bit format) and stays constant for all rounds. At run time, a device selects the dropout vector d that best corresponds to its resource availability. If resource availability changes at the device, the dropout vectors can be adjusted to these changes at almost zero overhead before every mini-batch. No weight copies, recompilation, repacking of weights, etc. are required for adapting the resource requirements.

Algorithm 1: Each Selected Device i (Client)

Require: D: LUT of Pareto-optimal dropout vectors (from DSE)
receive θ_{init} from server
$\theta \leftarrow \theta_{init}, c \leftarrow 0$
for each $b \in X_i$ **do** ▷ *iterate over mini-batches from local data*
$r \leftarrow r_i(t)$ ▷ *current resource availability*
$d \leftarrow D[r]$ ▷ *resource-aware dropout vector*
Update dropout values of local NN with d
$\theta \leftarrow \theta - \eta \frac{\partial}{\partial\theta} L(b;\theta)$ ▷ *update step*
$c \leftarrow c + \text{MACs}(d)$ ▷ *accumulate computations*
send $(\theta - \theta_{init}, c)$ to server ▷ *weight update and computations*

In an FL setting, each device selects its dropout vector at run time according to its resource availability, as shown in Algorithm 1. This is done at the granularity of single mini-batches, i.e., devices can quickly react to changes. Additionally, the server does not need to know the resource availability at each device at the beginning of the round, reducing signaling overhead, and avoiding the requirement to know resource availability ahead of time. This is important to maintain scalability with the number of devices. At the end of each round, the devices report back the weight updates and the computational resources they put into training (number of MACs, as stored in the LUT). The server (Algorithm 2) performs a weighted averaging of the received updates w.r.t. the devices' reported computational resources. Thereby, updates from devices that have trained with lower dropout rates, are weighted stronger. This is an extension of *FedAvg* (McMahan et al. 2017), which performs weighted averaging only based on the number of mini-batches. In the case of constant and same resource availability on all de-

Algorithm 2: Server

$\theta_0 \leftarrow$ random initialization
for each round $t = 1, 2, \ldots$ **do**
  $K \leftarrow$select n devices
  broadcast θ_{t-1} to selected devices K
  receive $(d\theta_i, c_i)$ from devices $i \in K$
  $C \leftarrow \sum_{i \in K} c_i$ ▷ *tot. computations*
  $d\Theta \leftarrow \sum_{i \in K} c_i \cdot d\theta_i$ ▷ *weighted sum*
  $d\theta \leftarrow d\Theta / C$ ▷ *weighted average*
  $\theta_t \leftarrow \theta_{t-1} + d\theta$

	FEMNIST	CIFAR-10	CIFAR-100
#Devices	3,550	100	100
#Samples/device	181±70.7	500	500
Devices/round	35	10	10
Resources var.	3×	4×	4×

Table 1: System configuration for FL.

vices, our coordination technique behaves the same as *FedAvg*. As we do not change the type of data exchanged between the devices and the server, compared to *FedAvg*, we can still apply and adopt techniques that mitigate communication aspects, such as compression and sketched updates (Shi et al. 2020).

Experimental Results

This section demonstrates the benefits of *DISTREAL* with heterogeneous resource availability in an FL system.

Experimental Setup We study synchronous FL as described in the system model. We report the classification accuracy of the synchronized model at the end of each round. Our main performance metric is the *convergence speed*, i.e., the accuracy achieved after a certain number of rounds, but we also report the final accuracy after convergence.

The three datasets used in our experiments are *Federated Extended MNIST* (FEMNIST) (Cohen et al. 2017) with non-independently and identically distributed (non-iid) split data, similar to LEAF (Caldas et al. 2019), and CIFAR-10/100 (Krizhevsky and Hinton 2009). FEMNIST consists of 641,828 training and 160,129 test examples, each a 28×28 grayscale image of one out of 62 classes (10 digits, 26 upper- and 26 lower-case letters). CIFAR-10 consists of 50,000 training and 10,000 test examples, each a 32×32 RGB image of one out of 10 classes such as airplane or frogs. CIFAR-100 is similar to CIFAR-10 but uses 100 classes. Table 1 summarizes the configurations.

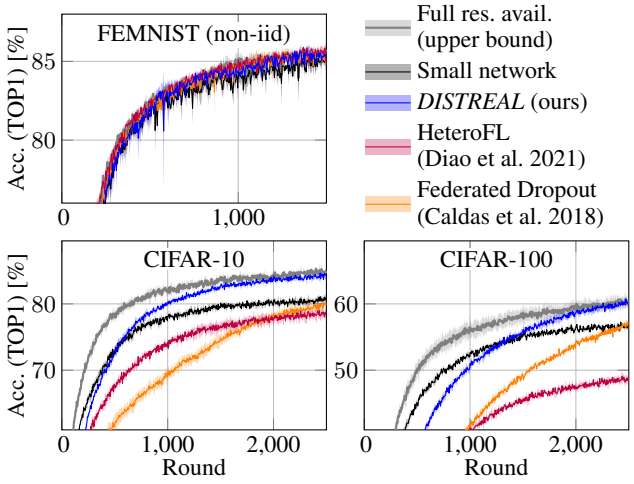
For FEMNIST, we use a similar network as used in Federated Dropout (Caldas et al. 2018), with a depth of 4 layers, requiring 4.0 million MACs in the forward pass. We use DenseNet (Huang et al. 2017) for CIFAR-10 and CIFAR-100 with growth rate $k = 12$ and depth of 40 and 100, respectively. This results in 74 million MACs for CIFAR-10 and 291 million MACs for CIFAR-100 in the forward pass. The DSE for these NNs takes around 15, 270, and 330 compute-hours, respectively, on a system with an Intel Core i5-4570 and an NVIDIA GeForce GTX 980. More details about the NN configurations and the computational complexity of the DSE are presented in the appendix.

We compare *DISTREAL* to four baselines:

1. **Full resource availability.** All devices have the full resources to train the full NN in each round. This is a theoretical baseline, which serves as an upper bound.
2. **Small network.** The NN complexity is reduced to fit the weakest device. Thereby, each device can train the full (reduced) NN in each round with FedAvg. For CIFAR-10 and CIFAR-100, we reduce the depth of DenseNet to 19 and 40, respectively. Because the network of FEMNIST already has only a few layers, we reduce the number of filters of the convolutional layers.
3. **Federated Dropout** as in (Caldas et al. 2018). Similar to our technique, it uses dropout to reduce the computational complexity. However, the same dropout rate is used for all layers. To have a fair comparison, we extend the technique of Caldas et al. (2018) to allow for different dropout rates for different devices according to the resource availability. The rates are determined by the server at the start of each round as in the original technique.
4. **HeteroFL** as in (Diao, Ding, and Tarokh 2021). It uses a shrinking ratio $0<s<1$. The NN is divided into several levels $p = 1, 2, \ldots$, where level p reduces the width of each hidden channel to a fraction s^{p-1}. This is done on the server at the beginning of each round. (Diao, Ding, and Tarokh 2021) provides no details on how to set s. We use s=0.7, as it shows the best performance.

Heterogeneity Across Devices We first study heterogeneity across devices, i.e., devices have different resource availability but for now, there are no changes over time. We select the resource availability at each device randomly and uniformly from a range with the upper bound being selected such that training the full NN without dropout is possible. The variability in the resource availability, i.e., ratio of upper to lower bound in the range, is reported in Table 1. We repeat every experiment three times and report the average and standard deviations of the classification accuracy.

Fig. 5 shows the accuracy results for the three datasets. FEMNIST uses a simple network. We observe that the small network baseline has the lowest convergence speed, with all other techniques showing similar performance. The varying quantity (see Table 1) and distribution of local data on the devices make the training noisy. Nevertheless, *DISTREAL* is not more sensitive to non-iid data than other solutions and reaches the same convergence speed and final accuracy. With CIFAR-10, *DISTREAL* achieves a significantly higher convergence speed than Federated Dropout or HeteroFL and reaches the accuracy of the theoretical baseline after 2,000 rounds. The simple baseline that uses a smaller network on all devices initially converges faster but saturates early. We also evaluate the final accuracy, where we train with each technique for 7,500 rounds. This ensures that all techniques have fully converged. *DISTREAL* and Federated Dropout reach the highest final accuracy, similar to the upper bound of full resource availability. Similar observations can be made with CIFAR-100.



(a) Convergence speed

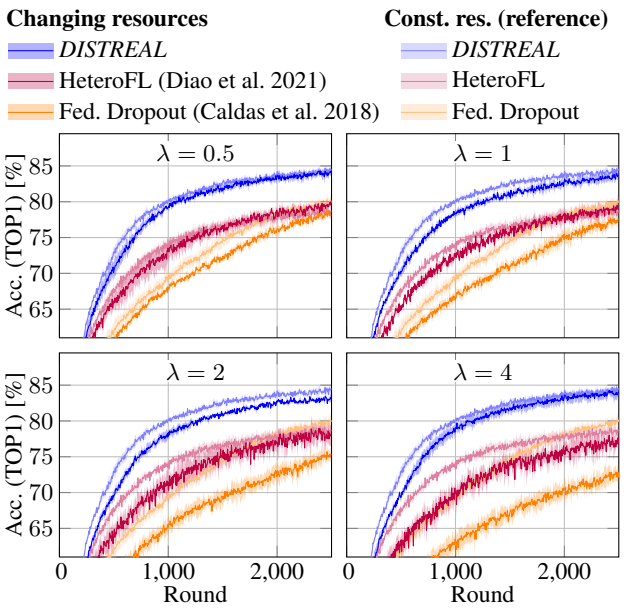
	FEMNIST	CIFAR-10	CIFAR-100
Full res. avail. (upper bound)	87.4±0.3	87.8±0.1	65.6±0.5
Small NN	86.4±0.1	82.2±0.6	57.8±0.5
DISTREAL	86.7±0.1	85.7±0.3	65.7±0.5
Diao 2021	87.1±0.4	81.2±0.5	52.2±0.6
Caldas 2018	86.9±0.3	84.2±0.3	65.3±0.5

(b) Final accuracy (after 7,500 rounds)

Figure 5: Convergence during FL on heterogeneous devices. *DISTREAL* improves the convergence speed, while still reaching the same or a higher final accuracy than others.

As the resources are not changing over time, the main contributions of *DISTREAL* in this scenario are the application of the DSE, which enable devices to efficiently utilize the available resources and the fact that *DISTREAL* applies a probabilistic approach and drops different filters in different mini-batches, allowing to support a large number of resource levels. It, therefore, outperforms Federated Dropout in terms of convergence speed, which uses the same dropout rates over all the layers, and HeteroFL, which supports only a few resource levels and removes the filters in always the same order, as *DISTREAL* fully utilizes the available resources, and uses these resources more efficiently w.r.t. convergence. Besides, deeper is the trained model, higher is the relative gain.

Heterogenity Across Devices and Over Time This section studies a fully heterogeneous FL system, i.e., resource availability varies between devices and for each device over time. As discussed in the introduction, this for instance due to shared resource contention between the learning task and other tasks. The available resources for learning may change at any time (workload changes happen in the order of seconds (Tu et al. 2014)), i.e., also in the middle of a round. As these changes stem from changes in the environment, they may be unpredictable to the device (Duc et al. 2019). We model them as random, with the time between changes following an exponential distribution with rate parameter λ.



(a) Convergence speed

	$\lambda = 0.5$	$\lambda = 1$	$\lambda = 2$	$\lambda = 4$
DISTREAL	86.3±0.3	86.0±0.2	85.1±0.3	85.9±0.4
Diao 2021	82.6±0.2	81.9±1.3	81.7±0.5	81.3±0.3
Caldas 2018	83.4±0.6	83.2±0.4	82.4±0.4	82.7±0.6

(b) Final accuracy (after 7,500 rounds)

Figure 6: Convergence with CIFAR-10 on heterogeneous devices where resources availability changes randomly over the time with varying rate parameter λ. *DISTREAL* achieves a higher convergence speed than the state of the art, and the highest final accuracy, independently of λ. Experiments with full resource availability or with a small network are not repeated as they perform the same as in Fig. 5.

The absolute resource availability levels are sampled from the same range as in the previous section, i.e., also according to Table 1. Thereby, the average resource availability across all devices and over time is the same as in the previous section. We study four different values of $\lambda \in \{0.5, 1, 2, 4\}$, to simulate a range of slowly to rapidly changing scenarios.

Fig. 6a shows the convergence speed for CIFAR-10. We also plot the convergence speed with constant resources (previous results from Fig. 5a) for reference. We observe that the convergence speed with *DISTREAL* is not dependent on the rate of resource changes and almost matches the results of the previous section. In contrast, the convergence speeds of HeteroFL and Federated Dropout significantly degrade with higher λ. In addition, *DISTREAL* reaches the highest final accuracy (Fig. 6b), independently of λ. The baselines with full resource availability and small model perform the same as in Fig. 5, therefore are not shown again.

HeteroFL and Federated Dropout both select the trained model subsets on devices at the server at the start of a round. The devices train on the assigned subset for the whole round

and hence cannot react to potential unpredictable changes in the resource availability during a round. An increase in resource availability results in underutilization of available resources, as training finishes early and the device is idle until the end of the round. A decrease in resource availability results in the training not finishing in time (i.e., the device becomes a straggler), leading to the device being dropped from the round. In contrast, *DISTREAL* adjusts the resource requirements of training at run-time by selecting a different dropout vector when a change occurs, finishing the training in time and fully utilizing the available resources.

These results show the importance of tackling the challenges discussed in the introduction, to *fully* and *efficiently* utilize available resources on each device. Our technique achieves this by performing dropout at the devices, enabling them to react fast to the changes in a fine-grained manner. This enables to fully utilize all available resources, making convergence robust to changes in the resource availability. Furthermore, the DSE enables us to efficiently utilize the available resources by finding Pareto-optimal dropout vectors w.r.t. resource requirements and achieved convergence speed. These gains in convergence speed do not come at the cost of a lower final accuracy.

Conclusion

We addressed the problem of distributed training of NNs under heterogeneous resource availability, proposing *DISTREAL*. Our results show that *DISTREAL* significantly improves the convergence speed in a heterogeneous FL system, where resources vary both between devices *and over time* without compromising on the final accuracy. This is as our solution provides each device the capability to adjust the training in a fine-grained manner, enabling it to fully and efficiently utilize its available, but limited, resources.

Acknowledgments

This work is in parts funded by the Deutsches Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research in Germany).

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.

Amir, M.; and Givargis, T. 2018. Priority Neuron: A Resource-Aware Neural Network for Cyber-Physical Systems. *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 37(11): 2732–2742.

Amodei, D.; Hernandez, D.; Sastry, G.; Clark, J.; Brockman, G.; and Sutskever, I. 2018. AI and Compute. https://openai.com/blog/ai-and-compute/. Accessed: 2020-09-29.

Ananthanarayanan, G.; Bahl, P.; Bodík, P.; Chintalapudi, K.; Philipose, M.; Ravindranath, L.; and Sinha, S. 2017. Real-Time Video Analytics: The Killer App for Edge Computing. *Computer*, 50(10): 58–67.

Baker, B.; Gupta, O.; Raskar, R.; and Naik, N. 2017. Accelerating Neural Architecture Search using Performance Prediction. *arXiv preprint arXiv:1705.10823*.

Bernardos, C. J.; and Uusitalo, M. A. 2021. European Vision for the 6G Network Ecosystem.

Biscani, F.; and Izzo, D. 2020. A Parallel Global Multiobjective Framework for Optimization: pagmo. *Journal of Open Source Software*, 5(53): 2338.

Bonawitz, K.; Eichner, H.; Grieskamp, W.; Huba, D.; Ingerman, A.; Ivanov, V.; Kiddon, C.; Konečný, J.; Mazzocchi, S.; McMahan, H. B.; Overveldt, T. V.; Petrou, D.; Ramage, D.; and Roselander, J. 2019. Towards Federated Learning at Scale: System Design. In *SysML Conference*.

Caldas, S.; Duddu, S. M. K.; Wu, P.; Li, T.; Konečnỳ, J.; McMahan, H. B.; Smith, V.; and Talwalkar, A. 2019. Leaf: A benchmark for federated settings. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Caldas, S.; Konečny, J.; McMahan, H. B.; and Talwalkar, A. 2018. Expanding the Reach of Federated Learning by Reducing Client Resource Requirements. *arXiv preprint arXiv:1812.07210*.

Chang, H.; Shejwalkar, V.; Shokri, R.; and Houmansadr, A. 2019. Cronus: Robust and Heterogeneous Collaborative Learning with Black-Box Knowledge Transfer. *arXiv preprint arXiv:1912.11279*.

Chen, Y.; Ning, Y.; Slawski, M.; and Rangwala, H. 2020a. Asynchronous Online Federated Learning for Edge Devices with Non-IID Data. In *International Conference on Big Data (Big Data)*. IEEE.

Chen, Y.; Qin, X.; Wang, J.; Yu, C.; and Gao, W. 2020b. FedHealth: A Federated Transfer Learning Framework for Wearable Healthcare. *IEEE Intelligent Systems*, 35(4).

Chen, Y.; Xie, Y.; Song, L.; Chen, F.; and Tang, T. 2020c. A Survey of Accelerator Architectures for Deep Neural Networks. *Engineering*, 6(3): 264–274.

Cohen, G.; Afshar, S.; Tapson, J.; and Van Schaik, A. 2017. EMNIST: Extending MNIST to Handwritten Letters. In *International Joint Conference on Neural Networks (IJCNN)*.

Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2): 182–197.

Dhar, S.; Guo, J.; Liu, J.; Tripathi, S.; Kurup, U.; and Shah, M. 2019. On-device Machine Learning: An Algorithms and Learning Theory Perspective. *arXiv preprint arXiv:1911.00623*.

Diao, E.; Ding, J.; and Tarokh, V. 2021. HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients. In *International Conference on Learning Representations (ICLR)*. IEEE.

Duc, T. L.; Leiva, R. G.; Casari, P.; and Östberg, P.-O. 2019. Machine Learning Methods for Reliable Resource Provisioning in Edge-Cloud Computing: A Survey. *ACM Computing Surveys (CSUR)*, 52(5).

Elsken, T.; Metzen, J. H.; and Hutter, F. 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research (JMLR)*, 20.

European Commission. 2021a. HORIZON-CL4-2022-DATA-01-02 – Cognitive Cloud: AI-enabled Computing Continuum from Cloud to Edge (RIA).

European Commission. 2021b. HORIZON-CL4-2022-DATA-01-03 – Programming Tools For Decentralised Intelligence and Swarms (RIA).

Graham, B.; Reizenstein, J.; and Robinson, L. 2015. Efficient Batchwise Dropout Training using Submatrices. *arXiv preprint arXiv:1502.02478*.

Horváth, S.; Laskaridis, S.; Almeida, M.; Leontiadis, I.; Venieris, S. I.; and Lane, N. D. 2021. FjORD: Fair and Accurate Federated Learning under heterogeneous targets with Ordered Dropout. *arXiv preprint arXiv:2102.13451*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely Connected Convolutional Networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Huang, G.; Sun, Y.; Liu, Z.; Sedra, D.; and Weinberger, K. Q. 2016. Deep Networks with Stochastic Depth. In *European Conference on Computer Vision (ECCV)*. Springer.

Imteaj, A.; Thakker, U.; Wang, S.; Li, J.; and Amini, M. H. 2020. Federated Learning for Resource-Constrained IoT Devices: Panoramas and State-of-the-art. *arXiv preprint arXiv:2002.10610*.

Krizhevsky, A.; and Hinton, G. 2009. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet Classification with Deep Convolutional Neural Networks. In *Neural Information Processing Systems (NIPS)*.

Li, C.; Wang, G.; Wang, B.; Liang, X.; Li, Z.; and Chang, X. 2021. Dynamic Slimmable Network. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Li, D.; and Wang, J. 2019. Fedmd: Heterogenous Federated Learning via Model Distillation. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Li, T.; Sahu, A. K.; Talwalkar, A.; and Smith, V. 2020a. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine*, 37(3): 50–60.

Li, T.; Sahu, A. K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; and Smith, V. 2020b. Federated Optimization in Heterogeneous Networks. In *Machine Learning and Systems (MLSys)*, volume 2.

Lin, T.; Kong, L.; Stich, S. U.; and Jaggi, M. 2020. Ensemble Distillation for Robust Model Fusion in Federated Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Liu, B.; Wang, L.; and Liu, M. 2019. Lifelong Federated Reinforcement Learning: A Learning Architecture for Navigation in Cloud Robotic Systems. *IEEE Robotics and Automation Letters*, 4(4): 4555–4562.

McMahan, H. B.; Moore, E.; Ramage, D.; Hampson, S.; et al. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*.

NetWorld. 2020. Smart Networks in the Context of NGI.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Neural Information Processing Systems (NeurIPS)*, 8024–8035.

Rapp, M.; Khalili, R.; and Henkel, J. 2020. Distributed Learning on Heterogeneous Resource-Constrained Devices. *arXiv preprint arXiv:2006.05403*.

Shi, Y.; Yang, K.; Jiang, T.; Zhang, J.; and Letaief, K. B. 2020. Communication-Efficient Edge AI: Algorithms and Systems. *IEEE Communications Surveys & Tutorials*, 22(4).

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T. P.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587): 484–489.

Song, Z.; Wang, R.; Ru, D.; Peng, Z.; Huang, H.; Zhao, H.; Liang, X.; and Jiang, L. 2019. Approximate Random Dropout for DNN Training Acceleration in GPGPU. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1): 1929–1958.

Tan, M.; and Le, Q. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning (ICML)*.

Tann, H.; Hashemi, S.; Bahar, R. I.; and Reda, S. 2016. Runtime Configurable Deep Neural Networks for Energy-Accuracy Trade-Off. In *Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. IEEE.

Thakker, U.; Beu, J.; Gope, D.; Zhou, C.; Fedorov, I.; Dasika, G.; and Mattina, M. 2019. Compressing RNNs for IoT Devices by 15-38x using Kronecker Products. *arXiv preprint arXiv:1906.02876*.

Tu, C.-H.; Hsu, H.-H.; Chen, J.-H.; Chen, C.-H.; and Hung, S.-H. 2014. Performance and Power Profiling for Emulated Android Systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 19(2).

Xie, C.; Koyejo, S.; and Gupta, I. 2020. Asynchronous Federated Optimization. In *Workshop on Optimization for Machine Learning*.

Xu, Z.; Yang, Z.; Xiong, J.; Yang, J.; and Chen, X. 2019. ELFISH: Resource-Aware Federated Learning on Heterogeneous Edge Devices. *arXiv preprint arXiv:1912.01684*.

Yang, T.; Andrew, G.; Eichner, H.; Sun, H.; Li, W.; Kong, N.; Ramage, D.; and Beaufays, F. 2018. Applied Federated Learning: Improving Google Keyboard Query Suggestions. *arXiv preprint arXiv:1812.02903*.

You, Y.; Zhang, Z.; Hsieh, C.-J.; Demmel, J.; and Keutzer, K. 2018. ImageNet Training in Minutes. In *International Conference on Parallel Processing (ICPP)*.

Yu, J.; Yang, L.; Xu, N.; Yang, J.; and Huang, T. 2018. Slimmable Neural Networks. *International Conference on Learning Representations (ICLR)*.

Yu, R.; and Li, P. 2021. Toward Resource-Efficient Federated Learning in Mobile Edge Computing. *IEEE Network*, 35(1): 148–155.

Zhang, C.; Patras, P.; and Haddadi, H. 2019. Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys and Tutorials*, 21(3): 2224–2287.

Zhang, H.; Ananthanarayanan, G.; Bodik, P.; Philipose, M.; Bahl, P.; and Freedman, M. J. 2017. Live Video Analytics at Scale With Approximation and Delay-Tolerance. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; and Zhang, J. 2019. Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing. *Proceedings of the IEEE*, 107(8): 1738–1762.