Federated Learning for Computationally Constrained Heterogeneous Devices: A Survey

KILIAN PFEIFFER and MARTIN RAPP, Karlsruhe Institute of Technology, Germany
RAMIN KHALILI, Munich Research Center Huawei Technologies, Germany
JÖRG HENKEL, Karlsruhe Institute of Technology, Germany

334

With an increasing number of smart devices like internet of things devices deployed in the field, offloading training of neural networks (NNs) to a central server becomes more and more infeasible. Recent efforts to improve users' privacy have led to on-device learning emerging as an alternative. However, a model trained only on a single device, using only local data, is unlikely to reach a high accuracy. Federated learning (FL) has been introduced as a solution, offering a privacy-preserving tradeoff between communication overhead and model accuracy by sharing knowledge between devices but disclosing the devices' private data. The applicability and the benefit of applying baseline FL are, however, limited in many relevant use cases due to the heterogeneity present in such environments. In this survey, we outline the heterogeneity challenges FL has to overcome to be widely applicable in real-world applications. We especially focus on the aspect of computation heterogeneity among the participating devices and provide a comprehensive overview of recent works on heterogeneity-aware FL. We discuss two groups: works that adapt the NN architecture and works that approach heterogeneity on a system level, covering Federated Averaging, distillation, and split learning–based approaches, as well as synchronous and asynchronous aggregation schemes.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Computing methodologies → Cooperation and coordination;**

Additional Key Words and Phrases: Machine learning, federated learning, resource-constraints, heterogeneous, distributed computing

ACM Reference format:
Kilian Pfeiffer, Martin Rapp, Ramin Khalili, and Jörg Henkel. 2023. Federated Learning for Computationally Constrained Heterogeneous Devices: A Survey. *ACM Comput. Surv.* 55, 14s, Article 334 (July 2023), 27 pages.
https://doi.org/10.1145/3596907

1 INTRODUCTION

In recent years, a paradigm shift in **machine learning (ML)** on smart devices, such as **internet of things (IoT)** or smartphones, could be observed. Previously, most deployments of ML solutions on such devices were designed to train the ML model once at design time in a high-performance

This work is in parts funded by the Deutsches Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research in Germany).
Authors' addresses: K. Pfeiffer, M. Rapp, and J. Henkel, Karlsruhe Institute of Technology, Karlsruhe, Germany; emails: {kilian.pfeiffer, martin.rapp, henkel}@kit.edu; R. Khalili, Munich Research Center Huawei Technologies, Munich, Germany; email: ramin.khalili@huawei.com.

0360-0300/2023/07-ART334 $15.00
https://doi.org/10.1145/3596907

cloud [83]. At runtime, the fully trained model gets deployed on the devices, where only inference tasks are performed. The increasing number of smart devices and recent hardware improvements open the possibility of performing (continuous) on-device learning. This paradigm shift is mainly motivated by privacy and security concerns and motivated by policies like the European Union's GDPR [30] or California's CCPA [11].

On-device learning has several advantages over centralized training. In the extreme case, when devices independently perform on-device learning, no information about local data samples leaves the device, maintaining users' privacy. Additionally, on-device learning eliminates communication that comes with uploading collected training samples to a centralized server. This is particularly relevant with the expected growth rate of IoT devices [69], each equipped with sensors producing massive amounts of data, where the increasing communication burden limits the ability to process all this data centrally. However, this comes with new challenges. First, not all deployed devices are capable of doing any training and hence rely on an externally trained model. Further, devices that are capable can only train on their own samples (a tiny subset of a potential centralized dataset), and the resulting models suffer from accuracy losses and weak generalization properties.

Federated learning (FL) [70] is a recently introduced decentralized approach, where training is done in a distributed manner on each device, but devices can still collaborate to share knowledge. Federated learning improves privacy [8, 96] compared to the traditional centralized cloud paradigm. At the same time, FL enables devices to exchange relevant knowledge, improving the models' ability to generalize and overall increasing the accuracy. There are several techniques for how knowledge can be exchanged. The most common approach is to exchange **neural network (NN)** model weights [70], but there are other methods, as will be discussed later.

While FL systems for smart devices are proposed for a lot of different fields like healthcare [19, 32, 49, 105], transportation [21, 62, 78, 84], and robotics [64, 65] using natural language processing, computer vision, and reinforcement learning policies, only a few production use cases like the **Google Keyboard (GBoard)** [100] provide evidence of the success of the FL approach. We argue that real-world applications powered by FL are challenging to build because of the heterogeneity present in these environments [7, 46, 60], as almost any real-world system has heterogeneous properties that affect the efficacy of an FL system. A key factor of heterogeneity is the devices' different capabilities to perform training of an NN due to different degrees of *computational resources.*[1] Training NNs is computationally expensive due to the high number of trained parameters and its iterative search for a solution. This manifests itself in long training times, ranging up to several weeks for complex tasks. Today's IoT devices, smartphones, and embedded systems are still heavily constrained in their training capabilities.

For example, the *PM2.5* [16, 17] IoT sensor network continuously measures air quality (fine particular matter below a diameter of 2.5 μm) to train a model for anomaly detection. Several factors affect the devices' computational resources. The two most important ones in this setting are as follows:

- The open source nature of the project allows for a variety of hardware realizations, hence sensor devices have heterogeneous computational resources.
- Sensor devices are deployed in various environments, like indoors, where the devices are continuously powered, or outdoors where energy harvesting is required, limiting energy for training in a heterogeneous manner.

[1]Section 3 gives a detailed overview over the types of computational resources and the sources and characteristics of heterogeneity in these resources.

These and other sources of heterogeneity need to be considered when designing an FL system for cooperative learning. However, research on FL on computationally constrained heterogeneous systems is still in its infancy.

For instance, in GBoard, incorporating devices with heterogeneous resources is circumvented by forcing a homogeneous setting and excluding devices that do not fit. The ML model is exclusively trained on high-end smartphones that are in an idle state and have at least 2 GB of memory. These limitations might play a minor role when having a billion participating devices, as it could be the case in *cross-device* FL [50], but in smaller-scale applications (i.e., *horizontal cross-silo* [50]), excluding a large number of devices from the training reduces the achievable accuracy and generalization of the model. More importantly, there are cross-device cases where excluding devices also excludes an essential share of data that is exclusively available on constrained devices [68]. Because of fairness or fair representation, it might be required to learn from these devices. Therefore, computation-heterogeneity-aware FL is required to enable FL to learn from all devices and utilize any data.

While we mainly focus on computation heterogeneity in FL, heterogeneity also manifests itself in other domains. Devices have different data distributions and quantities of samples available. Also, they could have different communication capabilities. For a wider use of FL systems, these heterogeneities should be taken into account.

1.1 Scope and Contribution

This survey studies FL under computation heterogeneity. We also briefly discuss other sources of heterogeneity, such as communication and data heterogeneity, when there is an overlap with computation, but we refer readers to recently published surveys, e.g., References [86] and [56], for a more detailed discussion. In this survey, we first provide an extensive analysis of the sources of heterogeneity in devices in various kinds of environments and the implications for cross-device and horizontal cross-silo FL. We then provide a thorough analysis of the state-of-the-art techniques that cope with heterogeneous computation capabilities during FL training. We focus on literature that tackles computation heterogeneity on two different levels and exclude techniques that improve the resource efficiency of devices through hardware design considerations, such as accelerators [5], as they are not specific to FL. For techniques that exclusively target inference, such as federated **neural architecture search (NAS)** techniques, we refer to Lui et al. [66].

The presented techniques are grouped into two major groups, namely techniques that tackle heterogeneity through the devices' NN architectures level and techniques that address heterogeneity on the system level. An additional fine-grained categorization is based on the employed FL paradigm (**Federated Averaging (FedAvg)**, *distillation*, and *split learning*) for NN architecture-level techniques and is based on whether system-level techniques employ synchronous or asynchronous aggregation. We present our taxonomy of the research on FL with computation heterogeneity in Section 3 and Table 3, where we outline the different computation-related challenges that come from real-world applications and present a selection of works that make notable contributions to computation heterogeneity aware FL. Finally, we conclude by outlining open problems and remaining challenges. In contrast to previous surveys [1, 6, 47, 51, 67, 98, 103] that cover certain aspects of device heterogeneity in FL, we provide the following novel contributions:

- We provide an up-to-date review of the state of the art and analyze many techniques that are not yet covered in existing surveys. The problem of computation heterogeneity in FL only very recently gained relevance with upcoming training-capable IoT devices and proposals

Table 1. Previous Surveys in FL That Partially Cover Computation Heterogeneity in FL

Survey	Focus	Techniques w/ comp. heterogeneity	Distillation
Abdulrahman et al. [1]	FL Overview	7	✗
Xu et al. [98]	Asynchronous FL	4	✗
Imteaj et al. [47]	FL for constrained IoT	10	✗
Khan et al. [51]	FL for IoT	3	✗
Lo et al. [67]	FL Engineering Aspects	6	✗
Yin et al. [103]	FL Overview	3	✗
Bellavista et al. [6]	FL Deployment	7	✗
Ours	Comp. heterogeneity in FL	35	✓

for 5G sensor networks. Therefore, FL techniques addressing computation heterogeneity only recently gained popularity. In particular, about 50% of our covered techniques were published in 2021/2022. Besides, existing surveys cover less than half of the papers we survey. Table 1 presents a comparison to other works.

- Existing literature treats computation resources in a very abstract way (e.g., only considering the training time of the number of multiply accumulate operations), neglecting the different kinds of computational resource limitations and their different implications. This is as their main focus is different (see Table 1). For the design of future real-world FL applications, these abstract metrics do not suffice, as they do not well reflect the variety of heterogeneity sources that affect a deployed FL application. In contrast, we distinguish between four different concepts, which are constraint types, heterogeneity types, the scale of the heterogeneity, and its granularity.
- In contrast to existing surveys, we also include distillation-based FL approaches. This novel knowledge aggregation technique potentially enables NN model-agnostic FL and therefore enables the use of custom-tailored NN models to better account for the devices' heterogeneous capabilities. We provide an in-depth description of how distillation-based FL approaches exchange knowledge, present seven different approaches that utilize distillation to cope with computation heterogeneity, and discuss their current limitations.

The remainder of this survey is structured as follows: First, in Section 2, we introduce the major baseline algorithm of FL, FedAvg, and recently introduced distillation and split learning approaches and their respective advantages and disadvantages. In Section 3, we analyze the different sources that enforce computation constraints on devices and discuss how that leads to computation heterogeneity in FL. In Section 4, state-of-the-art work addressing computation heterogeneity is discussed. Finally, Section 5 presents open problems and future directions.

2 BASICS OF FEDERATED LEARNING

2.1 Problem Formulation

Similarly to distributed **stochastic gradient descent (SGD)**, FL follows a *server–client* model, where client workers (devices) do training and communicate with a central server to share knowledge. The simplified case of FL aims to learn a model under the constraint that the training data are locally distributed among many devices. Therefore the goal is to minimize the following function by finding optimal NN weights w s.t.:

$$\min_{w} \; f(w) \; \text{ where } \; f(w) := \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} f_k(w), \tag{1}$$

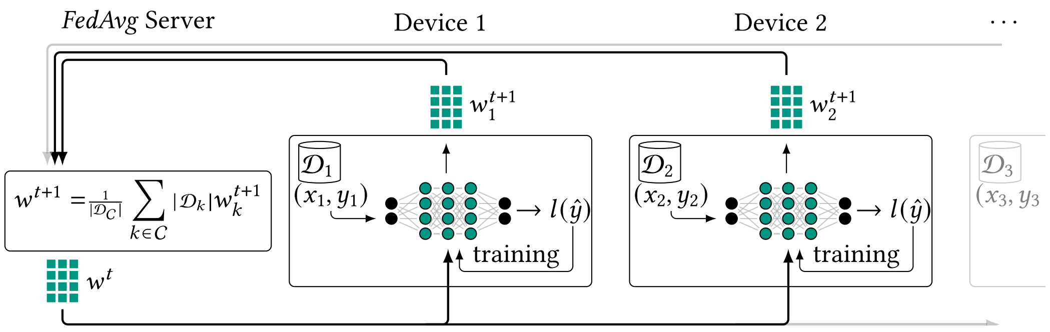

Fig. 1. FedAvg: Knowledge of devices $1, \ldots, K$ is shared through averaging the NNs' weights. Each device trains on its disjoint local data $\mathcal{D}_1, \ldots, \mathcal{D}_K$, producing newly trained weights $w_1, \ldots, w_K$. Every round, the new averaged weights w^t are distributed to all devices.

where $f(w)$ is the global loss (at a centralized server that handles knowledge aggregation) and $f_k(w)$ is the loss function of device k, where k is a device within the set $\mathcal{K}$. Each device exclusively has access to its local dataset $\mathcal{D}_k = \{x_k, y_k\}$, where x_k is the input and y_k is the label. The function $f_k(w)$, therefore, can be rewritten as

$$f_k(w) = l(x_k, y_k, w) \quad \{x_k, y_k\} \sim \mathcal{P} \ \forall k. \tag{2}$$

Each device k draws its samples from the distribution $\mathcal{P}$, resulting in $|\mathcal{K}|$ disjoint splits of the full dataset. The accuracy of such a scheme is bounded by the following two bounds. First, a natural upper accuracy bound is the centralized training case where a device has access to the whole dataset. The second is a device without any knowledge transfer, only relying on its local data, thus building a natural lower bound to the accuracy. We identify two main goals in computational heterogeneity-aware FL techniques: Increasing the convergence speed and reaching a high final accuracy despite having constrained devices.

2.2 Baseline Federated Averaging

FedAvg is an algorithm for FL that was first introduced by McMahan et al. [70] and is widely considered a baseline for FL. In the case of *synchronous* FedAvg, training is done in rounds. In each round, every device pulls the current model from the server. Now, each device trains for a fixed amount of mini-batches up to several epochs on its data. After training, each device uploads its models to the server. The server model is updated by averaging all the uploaded models. In the special case where during the local training phase, each device only applies one gradient step, FedAvg behaves like distributed SGD.

The following detailed description assumes synchronous round-based FedAvg. The aggregation scheme (one round) is visualized in Figure 1 and described in the following steps:

(1) At the beginning of every training round, the server deploys the current weights w^t on the set of devices. When starting the training $w^t = w^0$ to achieve the same random initialization of the NN on all devices. Since usually a very large number of devices participate in federated learning, a subset $C \subset \mathcal{K}$ of all devices is selected for training.
(2) Devices train their model (SGD steps) for a fixed number of mini-batches or epochs on their local data, consequently, every device does the same amount of training steps

$$w_k^{t+1} = w^t - \eta \nabla f_k(w^t), \tag{3}$$

where w_k^{t+1} is the resulting model weights set, while η is the learning rate.

(3) Afterward, devices upload their updated model weights w_k^{t+1} to the server.
(4) The server aggregates the devices' knowledge by averaging the received weights using

$$w^{t+1} = \frac{1}{|\mathcal{D}_C|} \sum_{k \in C} |\mathcal{D}_k| w_k^{t+1}, \tag{4}$$

where w^{t+1} represents the new global model and $|\mathcal{D}_C|$ the number of samples of devices in subset C. *The next round starts with Item 1 ($w^{t+1} \rightarrow w^t$).*

2.3 Client Selection in FedAvg

In Reference [70], as presented in Section 2.2, the availability of devices in $\mathcal{K}$ is modeled to be random, s.t. each round t a random subset $C^t \subset \mathcal{K}$ participates in FL. If a device k in the set C^t takes longer than others, then it delays the synchronous aggregation and, hence, slows down the overall FL training. If waiting becomes impractical, then the device has to be dropped from the current FL round, wasting its resources. Devices like these are called *stragglers*. Client selection techniques assume that device availability and their expected resources for training can be acquired by the server to select an optimal subset C^t of devices for each round. Further, they allow for variable per-round training time deadlines. Client selection techniques mainly target one or more of the following goals: Maximizing the overall FL convergence speed, minimizing the number of stragglers, or minimizing the overall energy spent on FL.

2.4 Asynchronous FedAvg

Alternatively, in *asynchronous* FedAvg, devices can pull the most recent model from the server at any time, perform local update steps on it, and upload it at any time. Knowledge aggregation is done at the server as soon as a new model update from a device arrives. *Stale devices*: In asynchronous schemes, devices cannot become stragglers, since their updated model can be aggregated instantly into the global model. Yet, devices that take too long to finish their training become stale devices. Stale devices upload their trained weights based on an old state of the global model, introducing instability. Chen et al. first provided evidence of that for asynchronous SGD [15]. Further, Xi et al., as well as Xu et al., demonstrate that staleness in FL lowers the convergence speed and affects the maximum reachable accuracy [97, 99].

2.5 Distillation for Federated Learning

Distillation techniques in FL take motivation from **knowledge distillation (KD)** [40], which was originally used to transfer knowledge of a larger NN into a reduced smaller one for model compression. Typically, NNs trained on classification problems output probabilities using a softmax layer that converts logits into probabilities. Knowledge distillation aims to distill the better generalization of larger models into smaller ones by not training the smaller network on the sample's class but rather on the sample's distribution (*soft label*) that is predicted by the larger network. Therefore, the smaller network not only learns the correct classes but also, through likelihood scores, learns about the larger model's knowledge representation.

Federated model distillation (FedMD) is an algorithm that uses distillation for FL that has been proposed recently by Li and Wang [59]. We describe how distillation is used in federated learning using FedMD: In addition to the devices' local (private) datasets, a second dataset $\mathcal{D}_p$ is introduced, which is a public dataset that is known to all participating devices. With consecutive training of private data and inference on public labels, devices transfer the knowledge of their private data into the public *soft labels.* The major difference to the previously discussed FedAvg scheme is that knowledge is not shared through model weights but through soft labels of a public dataset. This concept change allows for devices to be independent of the server model architecture.

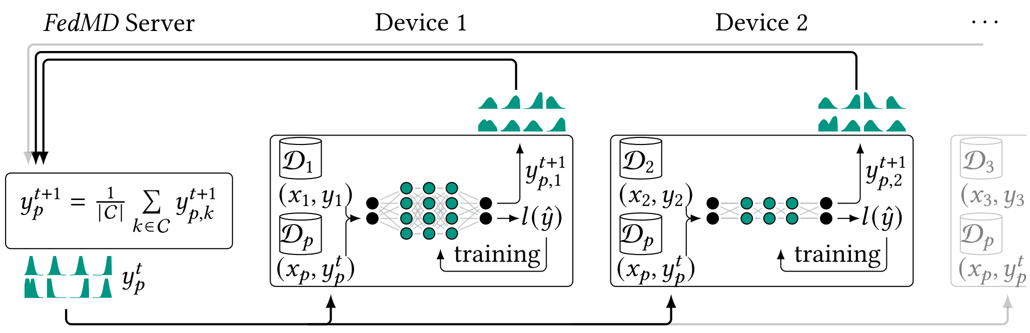

Fig. 2. FedMD: Knowledge of devices 1 to K is shared through soft labels y_p^t of the public dataset $\mathcal{D}_p$ that is known to all devices. Additionally, devices train on their private dataset $\mathcal{D}_1, \ldots, \mathcal{D}_K$. Devices transfer knowledge of their private data into public soft labels. Every round, the new averaged soft labels are distributed to the devices.

The steps of a synchronous knowledge aggregation round are visualized in Figure 2 and described in the following steps. In the general case, FedMD is applicable to any machine learning algorithm and task. For the sake of simplicity, we consider in the following detailed explanation only NNs that perform classification with a softmax activation layer. Again a subset C of all devices contributes in one round.

(1) At the beginning of each round, all devices download the current public dataset's soft labels y_p^t. In the first round, the public dataset's labels are one-hot encoded.
(2) First, all devices train their NNs on the public dataset $\mathcal{D}_p$, followed by training on their private data $\mathcal{D}_k = \{x_k, y_k\}$. After training on the private dataset, an inference phase on the public set follows. Instead of storing the one-hot encoded outputs, each device stores its generated soft label outputs $y_{p,k}^{t+1}$.
(3) The outputs, representing the probability distribution of the input over all possible classes, are uploaded for each public data sample to the server.
(4) The results from all participating devices are aggregated by using averaging

$$y_p^{t+1} = \frac{1}{|C|} \sum_{k \in C} y_{p,k}^{t+1}, \tag{5}$$

building a new averaged soft label for each sample in the public dataset. The public dataset is updated to $\{x_p, y_p^{t+1}\}$. *The round repeats with Item 1* $(y_p^{t+1} \rightarrow y_p^t)$.

2.6 Split Learning

Last, we also briefly elaborate on *split learning* techniques. In difference to FedAvg or distillation-based approaches, split learning techniques transfer information by using activations and gradients. In split learning, the NN model is split into two parts: A device model $a_k = f_k(w_k, x_k)$ for all devices $k \in \mathcal{K}$ and a single server model $f_s(w_s, a_k)$ that takes activations a_k from the devices as input. A combination of both can be used on device k for inference, s.t. $\hat{y}_k = f_s(w_s, f_k(w_k, x_k))$. Server and device models are trained in the following way:

(1) Device k calculates activations a_k by using $a_k = f_k(w_k, x_k)$ for its private data $\mathcal{D}_k$ and sends activations a_k with the respective class labels y_k to the server.
(2) The server receives the activations and class labels and calculates $\hat{y} = f_s(w_s, a_k)$ as well as the gradient $\nabla f_s(w_s)$. The gradient step (w.r.t f_s) is applied on the server the gradient w.r.t to a_k is sent back to device k.

Table 2. (a) Computational Capabilities (Float Performance, RAM) Vary across Different Device Classes and (b) the Resource Requirements (Number of MAC Operations, Memory Footprint) for Training of Several Established Image Classification NNs (Training with *PyTorch* [76], on *CIFAR10* [55] with Batch Size 32)

(a) Computational resources of end-devices

Device	**FLOPS**	**RAM**
MSP430 Ser.	10^5–10^6	0.5–66 KB
STM32F7 Ser. (Arm Cortex-M7)	$2 \cdot 10^8$	256–512 MB
Raspberry Pi Ser.	10^8–10^{10}	512 MB–8 GB
Low-End Smartphones	10^{10}–10^{11}	1–2 GB
Nvidia Jetson Nano	10^{11}–10^{12}	2–4 GB
High-End Smartphones	10^{11}–10^{12}	4–8 GB
Server GPUs	10^{13}–10^{14}	32–100 GB

(b) Resource requirements of NNs

ML Model	**# MACs**	**Memory**
	(Forward)	(Training)
LeNet	$6.7 \cdot 10^5$	0.5 GB
ResNet18	$5.6 \cdot 10^8$	0.8 GB
EfficientNet	$3.2 \cdot 10^7$	0.9 GB
MobileNetV2	$9.6 \cdot 10^7$	1.4 GB
ResNet152	$3.7 \cdot 10^9$	5.3 GB

(3) Device k uses the received gradient from the server to calculate $\nabla f_k(w_k)$. *The procedure repeats with Item 1.*

Split learning techniques can reach a high convergence speed and allows us to reduce the computational burden on the devices. This comes at the cost of a high communication volume.

3 COMPUTATION HETEROGENEITY IN FEDERATED LEARNING

This section covers different types of heterogeneity in devices and their effects on FL systems.

Our main focus is on *computation heterogeneity in existing devices*, i.e., different devices cooperating in an FL system differ in their capabilities to train NNs.

3.1 Computational Resources in End-Devices

Table 2(a) shows the computational resources of several common end device classes in terms of **floating-point operations per second (FLOPS)** and RAM. These devices range from ultra-low-power MSP430 microcontrollers to high-performance server GPUs. Table 2(b) shows the resource requirements of several well-known image classification NNs, in terms of the number of MAC operations in the forward pass and required memory for training a mini-batch of size 32. Note that the number of MAC operations for training a whole epoch would be 3–5 orders of magnitude larger, because training additionally requires a backward pass, which has around 2× the MACs of the forward pass [4] (plus eventual computations for a stateful optimizer), and a device has hundreds to thousands of local training samples to process in a round.

We observe that it is unrealistic to aim at training recent NN topologies on all devices. For instance, an MSP430-based embedded device by far does not provide sufficient memory. This is an important observation that puts a limit on how far into the embedded domain we can push FL, and it is much more realistic to employ powerful embedded devices such as Raspberry Pi, NVIDIA Jetson, or smartphones in FL systems. Nevertheless, there is large heterogeneity even within one type of device. For instance, the computational performance of smartphones varies between 10^{10} and 10^{12} FLOPS and 512 MB and 8 GB RAM. Heterogeneity greatly affects FL, because, clearly, not all devices can perform the same computations in each round, as will be discussed in more detail in Section 3.2.

In general, this heterogeneity may have many different sources:

- Different hardware or software generations of devices or device tiers may cooperate in one FL system. These can be, for instance, different smartphone generations or hardware revisions of IoT devices [17]. Different hardware generations may be equipped with different

ML accelerators that speed up training. Devices also can have different amounts of memory or storage capacities, limiting the devices' training capabilities.

- Degradation of components affects the available resources. The two famous examples are battery fading, where the energy capacity and peak power capabilities of a rechargeable battery reduce over time [26, 29], and circuit degradation, which reduces the switching speed of a circuit over time, reducing the achievable performance [92].
- The power/energy supply may be subject to variation. For example, the power supplied by a solar cell varies with solar irradiation, which both depends on local random weather effects (clouds) and regular cycles (day/night, summer/winter). Similar effects apply to other energy harvesting techniques [33].
- Ambient temperature affects the efficacy of cooling. This limits the thermally safe power dissipation and, thereby, also limits computation [38].
- Shared resource contention with other applications running on a device affects the available resources for training. ML model training often runs in parallel to other tasks on the same platform [31]. This leads to fast-changing degrees of contention in shared resources such as CPU time, memory, or energy.

3.2 Categorization of Constraints and Heterogeneity

We distinguish between two main categories of computation constraints, *hard constraints* and *soft constraints*, that cause different kinds of computation heterogeneity in an FL system, namely *heterogeneity across devices*, *over rounds*, and *over time*. These heterogeneities can have different *scales* and *granularities*.

Hard Constraints: These constraints prevent a device from training a given NN model. The most prominent example is limited memory. Despite considerate efforts to shrink the number of parameters in modern NNs [42, 89], model architectures like *MobileNetV2* [43] still have millions of parameters, which need to be kept in memory in high-precision (e.g., 32-bit floating-point) during training. In addition to the model parameters, also activations have to be kept in memory for applying backpropagation. This may easily accumulate to more than 1 GB of memory for training, as shown in Table 2(b). If a chosen NN architecture in an FL system exceeds the devices' memory capacity or the memory is not available due to resource contention, then the device cannot participate in the FL system.

Soft Constraints: These constraints allow for training a certain NN architecture but prevent the device from achieving its full training throughput (e.g., FLOPS). The computational capability is affected by several factors, such as the used micro-architecture, degradation of components, unstable power supplies, or shared resource contention. Constraints like these enforce slower training. For a device participating in an FL system, soft constraints can prevent the device from finishing its local training on time, making it a straggler or a stale device.

The aforementioned constraints may be heterogeneous throughout the set of devices participating in FL and over the training duration. We differentiate between three types of heterogeneity caused by device constraints.

Heterogeneity across Devices: First, devices participating in an FL system may have different kinds of hard and soft constraints, causing heterogeneity across the devices. These kinds of constraints (e.g., availability of accelerator or memory capacity) are either known at design time or before starting the training and do not change over time. Different devices may be subject to different constraints that limit the training throughput. An example of heterogeneity across devices is a smartphone application FL system. As discussed above, a low-end smartphone operates only with 1/100th of the peak performance and may have only 1/8th of the memory capacity of a high-end smartphone. Table 2 also lists other devices and their respective training throughput and memory resources. To support heterogeneity across devices, an FL algorithm needs to be able to use different amounts of resources on different devices in a round.

Heterogeneity over Rounds: If the number of devices in an FL system is comparatively small (cross-silo FL), then devices need to participate in many FL rounds. This is, for instance, the case in robotics [45] or lifelong learning [64]. Soft constraints of devices may be determined by the devices' environments and change over the rounds. Examples of such constraints are the devices' expected battery level during training, the current power supply, or ambient temperatures. This includes all constraints that change slower than FL rounds (in the range of minutes to hours), can be predicted and are known prior to an FL round. To support heterogeneity over rounds, an FL algorithm needs to either implement stateless clients or explicitly support changes in the availability of the resources of a device.

Heterogeneity over Time: Soft constraints that change at a finer granularity than FL rounds cause *heterogeneity over time* in an FL system. These throughput changes cannot be predicted and randomly occur (in the range of milliseconds/seconds). One example is resource contention on a smartphone, where the share of available resources is unpredictable for the FL system and changes within seconds, i.e., much faster than FL rounds. To support heterogeneity over time, an FL algorithm needs to adaptively adjust the required resources at the client during the round without relying on the server.

In any real system, a combination of the different types of heterogeneities is expected to occur. Finally, there are two additional properties of the heterogeneity that describe the statistical distribution of constraints present in a set of devices.

Scale of Heterogeneity: Heterogeneity across devices or over rounds/time can have different *scales*. This scale can vary depending on the source of constraints. In a smartphone application with different tiers or hardware generations, memory capacity and peak performance can vary by a factor of 10× and 8×. Contrary to that, constraints caused by an aging effect (battery) may result in a peak power reduction of more than 50% [26], which translates to a throughput difference of around 20%, assuming a cubic relationship between dynamic power and performance (V^2f-scaling [37]). This is much lower than the scale of 10× observed with different tiers or hardware generations. An FL algorithm must support the scale of heterogeneity present in the system.

Granularity of Heterogeneity: The heterogeneity present in an FL system can have different *granularities*. In a smartphone application, where devices are equipped with different memory capacities (e.g., 1, 2, and 4 GB), an FL system has to account for only a small finite number of types of devices. However, devices experiencing resource contention can have a continuous range of total training throughput in an FL round. Either an FL algorithm supports arbitrary resource availability levels or quantizing the continuous range into a reasonable finite number of constraint levels is required. The number of supported levels by the algorithm must be high enough to avoid wasting too many resources, which can slow down the overall training. For example, resource contention (e.g., four other applications sharing CPU time) would reduce the available resources for training by 5×. If the FL algorithm supports only 5 levels, which is not uncommon, as we observe in Section 4, then the ratio between subsequent levels is at least 1.5× if levels are distributed exponentially. As an example, levels of [1×, 1.5×, 2.2×, 3.3×, 5×] could lead to 33% of available resources for training being wasted. Note that this gap increases strongly if a larger scale needs to be supported.

As with all the other properties, an FL technique must be able to cope with the granularity of the system at hand.

These different types of computational heterogeneity require different solutions. We analyze the capabilities of the state-of-the-art techniques for heterogeneity-aware FL to cope with all the different types of computational heterogeneity in Section 4.

3.3 Communication Heterogeneity

Knowledge transfer, for instance, through sharing NN parameters between devices, is only possible via communication. The throughput, latency, and reliability of communication channels are limited and can vary between devices causing stragglers or stale devices. Most of the current research focuses on making the transmission more efficient by using compression and quantization schemes [3, 28, 54, 72]. Even though we do not focus on communication heterogeneity in this survey, we nevertheless cover certain aspects of communication because of the inter-dependencies between communication and computation:

- The complexity of the trained NN architecture correlates with the model size that has to be transmitted to the server. Hence, reducing the NN structure (e.g., by pruning [48]) reduces not only the computation complexity but also the communication burden.
- Increasing training throughput and communication throughput can create a tradeoff scenario [81], because ultimately, both may compete for resources like energy [73].

3.4 Data Quantity Heterogeneity

In a real-world scenario, the quantity of the data gathered on different devices may vary. However, to guarantee a high accuracy, the model needs to be trained over all available data. This imposes more training overhead on devices with a higher quantity of data, as they need to perform higher numbers of mini-batch training per round. Techniques that cope with limited throughput (soft constraints) can be applied to such devices to prevent them from becoming a straggler or stale devices.

4 COMPUTATION HETEROGENEITY-AWARE FEDERATED LEARNING

4.1 Categorization of Techniques

We categorize techniques addressing FL with heterogeneous computational capabilities into two groups.

NN architecture level: We distinguish between techniques where devices can choose from a limited set of model architectures or submodels. These techniques stem from FedAvg and are covered in Section 4.2. Other techniques do not impose any limitations on the model architecture. These techniques build on top of FedMD and are covered in Section 4.3. Last, we cover split learning–based techniques in Section 4.4.

System level: In this case, heterogeneity is not addressed by varying model complexity but by allowing for variable-length rounds, grouping, and partial updates. Besides synchronous solutions, variable training times can also be accounted for by allowing for asynchronous updates. These approaches are covered in Section 4.5 and 4.6, respectively.

Last, we discuss which types of computation heterogeneity are addressed. All discussed techniques are listed in Table 3.

4.2 NN Architecture Heterogeneity Based on *FedAvg*

The following techniques adapt FedAvg to achieve model architecture heterogeneity. Allowing for variable model complexity in FedAvg is not straightforward, since the aggregation scheme relies on averaging of model weights. If the model architectures vary, then it is not clear how to match the parameters for averaging. Even networks with the same architecture can have different learned structures; thus, averaging their weights hurts performance [70]. One reason for that is the NNs' permutation invariance, which means that even two-layer networks trained on the same distribution can have different internal structures. Several research works, therefore, focus on matching internal features. Wang et al. [94] introduce *FedMA* to match layerwise filters together.

Table 3. Comparison of Techniques Aiming at Computation Heterogeneity-aware FL

Work	**Layer**	**Mechanism**		**Constraints**		**Heterogen.**		
		Async	**Knowledge ex.**	**Hard**	**Soft**	**D**	**R**	**T**
Section 4.2 NN Architecture Heterogeneity based on *FedAvg*								
Caldas et al. [9]	Model	—	Parameters	✓	✓	—	—	—
ELFISH Xu et al. [99]	Model	—	Parameters	✓	✓	✓	✓	—
DISTREAL Rapp et al. [80]	Model	—	Parameters	—	✓	✓	✓	✓
HeteroFL Diao et al. [25]	Model	—	Parameters	✓	✓	✓	✓	—
MFL Yu and Li [104]	Model	—	Parameters	✓	✓	✓	✓	—
FjORD Horváth et al. [41]	Model	—	Parameters	✓	✓	✓	✓	✓
FedRolex Alam et al. [2]	Model	—	Parameters	✓	✓	✓	✓	—
FedorAS Dudziak et al. [27]	Model	—	Parameters	✓	✓	✓	✓	✓
FedHM Yao et al. [101]	Model	—	Parameters	✓	✓	✓	✓	—
FLANC Mei et al. [71]	Model	—	Parameters	✓	✓	✓	✓	—
ZeroFL Qui et al. [79]	Model	—	Parameters	✓	✓	—	—	—
CoCoFL Pfeiffer et al. [77]	Model	—	Parameters	✓	✓	✓	✓	—
Section 4.3 NN Architecture Heterogeneity based on Distillation								
FedMD Li and Wang [59]	Model	—	Soft labels (public)	✓	✓	✓	—	—
Cronus Chang et al. [14]	Model	—	Soft labels (public)	✓	✓	✓	—	—
FedHE Hin et al. [39]	Model	✓	Soft labels (per class)	✓	✓	✓	✓	✓
FedProto Tan et al. [90]	Model	—	Soft labels (per class)	✓	✓	✓	—	—
FedDF Lin et al. [63]	Model	—	Soft labels (server)	✓	✓	✓	—	—
FML Shen et al. [85]	Model	—	Soft labels (public)	✓	✓	✓	—	—
Section 4.4 NN Architecture Heterogeneity based on Other Techniques								
FedGTK He et al. [85]	Model	✓	Soft labels & activ.	✓	✓	—	—	—
AdaSplit Chopra et al. [20]	Model	✓	Gradients & activ.	✓	✓	✓	✓	✓
Section 4.5 System Level Awareness Through Client Selection and Flexible Aggregation								
FedCS Nishio and Yonetani [75]	System	—	Parameters	—	✓	✓	—	—
TiFL Chai et al. [12]	System	—	Parameters	—	✓	✓	✓	—
FLANP Reisizadeh et al. [82]	System	—	Parameters	—	✓	✓	—	—
Oort Lai et al. [57]	System	—	Parameters	—	✓	✓	✓	—
PyramidFL Li et al. [58]	System	—	Parameters	—	✓	✓	✓	—
FedProx Li et al. [61]	System	—	Parameters	—	✓	✓	✓	✓
Wang et al. [93]	System	—	Parameters	—	✓	✓	✓	—
Wang et al. [95]	System	—	Parameters	—	✓	—	✓	—
Tran et al. [91]	System	—	Parameters	—	✓	✓	—	—
AutoFL Kim and Wu [52]	System	—	Parameters	—	✓	✓	✓	—
Section 4.6 System Level Awareness Through Asynchronous Aggregation								
ASO Chen et al. [18]	System	✓	Parameters	—	✓	✓	✓	✓
FedAsync Xie et al. [97]	System	✓	Parameters	—	✓	✓	✓	✓
Sprague et al. [87]	System	✓	Parameters	—	✓	✓	✓	✓
Papaya Huba et al. [44]	System	✓	Parameters	—	✓	✓	✓	✓
FedAT Chai et al. [13]	System	✓	Parameters	—	✓	✓	✓	✓

We differentiate between hard and soft constraints, and heterogeneity across devices (D), over rounds (R), and over time (T).

Similarly, Yurochkin et al. [106] focus on matching neurons by identifying similar neuron subsets to match features in non-**independent and identically distributed (iid)** data scenarios. Reliable feature matching would allow for combining networks with varying architectures as well as a better generalization in non-iid data scenarios. Current techniques circumvent direct feature matching of varying NN structures by training with regularly changing variable-sized dropout masks or by training subsets of the full network, as discussed in the following. These techniques share some similarities with well-known inference approaches like pruning. The major differences are that pruning is primarily done after training has converged, intending to create an efficient model for inference. Meanwhile, in the following approaches, smaller subsets remain embedded in the full-size NN architecture, while each part is constantly updated every round.

Unstructured Subsets: Caldas et al. [9] were within the first to address the high computation burden of FL, introducing **Federated Dropout (FD)** for FedAvg, where instead of the full network, only a subset of the network is trained and updated in every round. A fixed set of weights is set to zero, and the remaining weights are packed into a dense matrix for efficient computation. For convolutional layers, full filters are dropped. In their experiments, they achieve a 1.7× reduction of computations on *MNIST* [23] without hurting final accuracy. The provided results suggest that averaging subsets through dropout masks does not negatively impact the aggregation mechanism of FedAvg. While reducing the devices' computation effort, FD forces a fixed dropout rate on all devices, thus limiting the ability to adapt to heterogeneity across devices. Heterogeneity across devices is tackled by Xu et al. [99] in *ELFISH*, a method that identifies neurons that contribute much to convergence and builds dropout masks based on this information. Each device receives a specific dropout mask to best match its current computing capabilities. Masks are updated every round. In *DISTREAL* [80], the authors explore how subsets can be trained in environments with time-varying computational resources that change faster than FL rounds and are not known in advance. A mini-batch level granularity for training subsets by randomly switching filters of the **convolutional neural network (CNN)** during training is achieved. Additionally, DISTREAL does not require a common fixed subset ratio per layer. Instead, this design space is explored with a genetic algorithm to find Pareto-optimal per-layer subset ratios. The results show that in scenarios with fast-changing resources, randomly switching between filters and optimized per-layer subset ratios result in faster convergence and higher final accuracies compared to FD.

Structured Subsets: While FD and ELFISH utilize unstructured subsets (masks), where the subset parameters are scattered over the full-size NN structure, some other studies propose a strictly hierarchical nesting of subsets. *HeteroFL* is presented by Diao et al. [25], a FedAvg adaption that allows devices to select from specific subsets of the full model. A smaller set is constructed as a subset of the next bigger set, giving devices a hierarchical selection of networks. Aggregation is done by only averaging trained parameters from the devices. Therefore, some parts of the model are only updated by strong devices. Similarly, Yu and Li [104] propose partitioning of CNN layer width, depth, and kernel size by slices of power of two and introduce a submodel search algorithm to best match a submodel to the devices' individual resources. They only provide proof-of-concept experiments for small networks in homogeneous cases. Horváth et al. [41] present *FjORD*, where devices receive submodels of various complexities through applying **Ordered Dropout (OD)**. Differently to HeteroFL, in each local round, every device uniformly selects from different complexity levels (within its capabilities) for a short training period. Since higher-performance devices are not fully utilized this way, the authors apply KD on top of OD to transfer knowledge from larger complexity models to smaller ones during local training. For comparison, they extend FD with variable dropout rates for each device and show that their structured subsets outperform random dropout masks.

Hybrid Subsets: Also, a mixture of both approaches, specifically the use of structured but not hierarchical subsets, is considered. In difference to the previous approaches, this allows training of NN-models that exceed the strongest devices' capacity. Additionally, each parameter gets eventually trained by each device. The use of a rolling window approach is proposed by Alam et al. [2] in *FedRolex*. In each training round, a device trains a different slice of the NN. Federated NAS [35, 74, 102] techniques are proposed, using subsets of shared common structure to allow for device personalizing, aiming for better accuracy in non-iid cases and efficient models for inference. An advantage compared to previous techniques is that the devices' NN models can be independently optimized for their hardware; however, this comes at the cost of exploring the architecture search space, which can be resource hungry. Dudziak et al. [27] present *FedorAS*, a federated NAS technique that also accounts for device heterogeneity during training. Similarly to FedRolex, devices receive a subset of the full server model and, depending on their resources, switch between further splits of the subset on a mini-batch level. The common training is followed by an architecture search for several tiers based on the full model and, last, a fine-tuning step.

Low-Rank Factorization: Low-rank factorization is considered to select the subset's parameters. These techniques also root from inference compression. The major difference to its use for inference is that here the low-rank NN is updated during training and the low-rank updates are applied to the full model on the server. Yao et al. [101] present *FedHM*, where they create low-complexity submodels on the server by doing a low-rank factorization of the full model. Layer parameters with dimensions $m \times n$ are decomposed into two matrices with dimensions $m \times r$ and $r \times n$. The complexity of the model can be controlled via the rank r. Computationally weak devices perform two lightweight convolution operations based on the matrix decomposition instead of one complex operation. To avoid a strong accuracy degradation, the complexity reduction through matrix decomposition is preferably applied in the later layers of the NN model. The authors show that subsets through low-rank factorization can achieve higher accuracies than straightforward splitting but, more importantly, dramatically reduces the communication burden. A similar technique is employed by Mei et al. [71] in *FLANC*, where in the factorization, the parameters are decomposed in matrices with dimensions smaller than m and n, allowing for a reduction of the activation size and, hence, memory requirements.

Others: In *ZeroFL* [79], dropout masks in combination with sparse convolutions are used to lower the computational complexity in training (FLOPS) and reduce the communication volume, although special hardware and software support is required to enable real-world gains. Last, in *CoCoFL* [77], a technique is presented that does not use subsets of an NN for training. Instead, only for some layers per round gradients get calculated while the remainder of the layers are frozen. This saves computation time (fewer backpropagation components) and reduces the upload volume, since only updated layers must be uploaded. Further, the freezing of the remaining layers allows for the reduction of the computation complexity by using inference techniques, such as batch norm folding and quantization (e.g., `int8` instead of `float32`), while in width scaling approaches (HeteroFL), gains in computation time and communication volume are tightly coupled, and selective freezing and training can decouple those properties. The results show benefits over HeteroFL, especially in scenarios where the devices' constraints regarding computation and communication are decoupled.

Discussion: Most presented techniques send lower complexity subsets of the full NN model to the devices. Allowing flexible NN models addresses hard constraints (e.g., devices can train a submodel with a lower memory footprint). Approaches partly require the resources to be known prior to the round, limiting real-world use cases. A remaining challenge is the scale and granularity of heterogeneity: For example, HeteroFL uses five subsets and scales down the parameters exponentially down to a 250× reduction. Consequently, there is a 4× gap between the full model and the largest sub-model. It remains unclear if subsets maintain effectiveness under both large scale and high granularity (many subsets). Last, while devices are enabled to participate in training with a 250× reduction of parameters, it remains untested if these devices can make a meaningful contribution to the global model.

4.3 NN Architecture Heterogeneity Based on Distillation

Contrary to FedAvg, FedMD does not transfer knowledge by sharing model weights but by sharing soft labels of a public dataset. Since the aforementioned problems of FedAvg (matching of NN features) do not apply here, the devices' capabilities can be better matched with tailored models (even disclosed from the server) as long as they share the soft label representation at the NNs' output. This allows to address hard constraints. However, creating a suitable public dataset and distributing it to all devices may be challenging in real-world scenarios. Additionally, training and inference on public data are computationally expensive. Also, distributing the public dataset to all devices may induce a large communication volume, thereby increasing the communication overhead. Also, if knowledge is shared purely through soft labels, then participating devices have to be stateful. Consequently, when a new device joins the system at a later stage, it still needs to train its model from scratch with the public soft labels. This is in contrast to FedAvg, where new devices simply download the latest model weights and immediately benefit from the already performed training of other devices.

Distillation With Public Data: Li and Wang [59] introduce FedMD (in detail shown in Section 2.5), which utilizes KD for FL, directly addressing the heterogeneous computational capabilities of devices. They test their solution with the *EMNIST/MNIST* [22] and the *CIFAR10/100* [55] dataset (public/private) using 10 devices, each deploying a unique NN architecture. They achieve a 20% increase in accuracy on every device compared to an isolated (on-device) setting. Chang et al. [14] present *Cronus*, which is similar to FedMD and also allows for heterogeneous architectures. While in FedMD, public and private data are trained consecutively, Cronus mixes both for local training.

Mixture of Distillation and FedAvg: As distillation approaches lack behind FedAvg w.r.t. achievable accuracy, a mixture of both approaches is proposed. Lin et al. [63] present *FedDF*, which moves KD from the devices to the server, thus, removing the additional public dataset training and inference effort. FedDF, similarly to FedMD or Cronus, also allows for heterogeneous architectures, while here, the server is fully aware of the devices' architectures. Aggregation is done by averaging all devices' weights (similar to FedAvg) within groups, and building an averaged starting model for each group. Each averaged group model now acquires knowledge from averaged soft labels computed with all received devices' weights. Compared to FedAvg, FedDF shows better robustness in non-iid data cases and allows for more local steps between rounds without degrading the performance. A disadvantage of this approach is that it requires data for distillation on the server. Shen et al. [85] propose *FML*, where two models are deployed on each device. The first one is a custom model that best fits the devices' computational capabilities and data. The second one is a knowledge transfer model that is used with KD to transfer knowledge between both networks in both directions. FedAvg is used to average the weights of the knowledge transfer model on a server. While they outperform FedAvg and FedProx [61] in certain experiments, this approach comes with the major downside of an additional computational burden of knowledge sharing on the device and forcing a fixed architecture for knowledge sharing on all devices.

Single per-Class Representations: Transferring soft labels of a public dataset comes with a large computational burden. To address this, the transfer of single per-class representations is discussed. Hin and Edith present *FedHe* [39], which, similarly to previous approaches, allows for different model architectures per device. Contrary to *FedMD* or Cronus, FedHe does not use a public dataset for distillation. The devices' models only share a single per-class representation (soft label) of their output layer trained on private data with the server. On the server, the per-class representation is averaged asynchronously. The devices train on their private data with a mixture of one-hot and soft label loss. FedHe is, therefore, lightweight in communication and requires no training with public data. The authors show that FedHe outperforms FedMD in many scenarios. Similarly, Tan et al. present *FedProto* [90], where knowledge is exchanged with class prototypes instead of public soft labels. Contrary to FedHe, not the output representations of the classes are used but an internal representation (an intermediate layer output before the NN network's last layer) to allow for higher expressiveness. The models on the devices are trained with a combined loss that accounts for the one-hot encoded private data and normalizes the model by keeping representations of private samples close to the global class representations.

Discussion: Presented approaches provide the most flexibility for devices to adapt their model architecture (addressing hard constraints), therefore also achieving a finer granularity of the heterogeneity compared to using subsets. In cases where certain devices have certain ML accelerators for specific tasks or very small memory budgets, distillation-based approaches allow for specifically tailored NNs to be deployed on the devices. Still, distillation-based approaches show certain disadvantages. First, in most cases, they do not reach the same accuracy as FedAvg-based approaches. Second, if knowledge between server and devices is exchanged through soft labels, then devices are stateful. Consequently, it is required that each device participates in a high number of rounds to have a sufficiently trained local model. Therefore, they do not scale as well as FedAvg-based techniques w.r.t. the number of participating devices and are best suited for horizontal cross-silo scenarios. Last, distillation adds computational overhead, limiting the applicability for purely throughput-constrained devices.

4.4 NN Architecture Heterogeneity Based on Other Techniques

He et al. [34] present a combination of split learning and distillation in *FedGKT*, where two models are employed. A lightweight feature extractor on the devices to lower the computational burden and a more complex server model. Knowledge is shared in both directions: The server receives feature maps and respective soft labels from the devices. The devices receive soft labels from the server. However, FedGKT does not allow for heterogeneous splits between device and server and provides no aggregation algorithm that supports heterogeneity. The final full model is a combination of the device and server models. The knowledge exchange is done asynchronously to avoid stragglers. Chopra et al. enable device heterogeneity in split learning, where the full NN is split, and parts of the model are trained on the devices while the other part is trained on the server. They present *AdaSplit* [20], which allows for different device model sizes by varying the split position between the device and the server. While in baseline split learning, activations have to be uploaded to the server, and gradients have to be downloaded from the server, AdaSplit mitigates this by using a contrastive loss to train locally without server interaction and send activations to the server only after the local phase. The implementation allows for asynchronous transfer of gradients.

A downside of the approach is that it requires training the model partly on the server, i.e., computation of gradients by using the received activations, which is more complex than averaging and might show problems with scaling to many devices. Additionally, split learning techniques result in stateful devices, as every device's final model is a combination of device and server mode. Consequently, the presented techniques are best suited for horizontal cross-silo FL, as each device has to participate in many rounds to reach a sufficiently trained device model.

4.5 System Level Awareness through Client Selection and Flexible Aggregation

In heterogeneous FL systems, devices with soft constraints delay the parameter aggregation, since the server has to wait for the slowest device (straggler) [99]. To demonstrate the different behaviors over time FedAvg with stragglers is visualized in Figure 3. Additionally, asynchronous

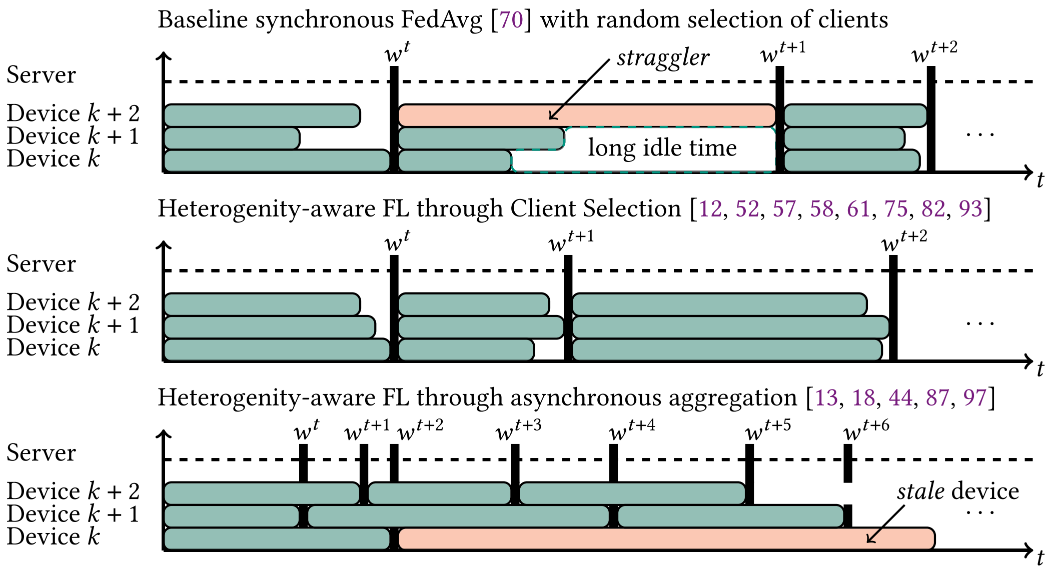

Fig. 3. FL aggregation of three different strategies over time. The first (top) is baseline FedAvg, where constrained devices can become stragglers, slowing down the rounds. The second (middle) is FL with system-level awareness (e.g., through client selection). The third (bottom) is asynchronous aggregation, where constrained devices can upload stale updates, hurting accuracy.

aggregation and aggregation with system awareness are displayed. The following works account for heterogeneity by minimizing straggler time through system awareness while still maintaining synchronous rounds.

Tier-based Client Selection: Nishio and Yonetani [75] propose *FedCS*, where they jointly consider the communication and computation resources of devices during device selection. At the beginning of each global round, the server requests the devices' current capabilities and selects a subset for the next aggregation step. The objective of this work is to achieve the highest accuracy in a limited time. Achieving this goal requires a tradeoff between maximum local training time (the longer the maximum round time is, the more devices can participate) and finishing many rounds within the given time budget. Chai et al. [12] introduce a tier-based aggregation scheme *TiFL*, where devices are profiled and grouped in tiers based on the time it takes to train for one epoch. Profiling can be done either static (prior to training) or continuously. The article discusses several tier selection schemes for training. Their experimental setup consists of five groups, where a higher tier always has the double performance of the previous tier. Training only with the fastest tier drastically reduces training time but hurts final accuracy, while uniform selection matches baseline FedAvg accuracy and reduces the training time by 50% (FedAvg has to wait for stragglers). They propose an adaptive selection to address non-iid data scenarios that selects tiers that have low accuracy with a higher probability, thereby achieving higher accuracy than baseline FedAvg, while significantly reducing the training time. Reisizadeh et al. [82] introduce *FLANP*, where the straggler problem is mitigated by utilizing the fast devices at the start. The authors assume an iid data environment and train with fast devices until they reach an accuracy threshold. They increase the accuracy threshold and iteratively add more (slower) devices to training. This slows down training but increases accuracy, since, every round, more data are available. Using this technique, FLANP converges a lot faster compared to FedAvg when having heterogeneous devices.

Presented techniques rely on similarities between devices, e.g., a low number of groups per round. An increase in the granularity and the scale of the heterogeneity deteriorates the gains. While an active selection of devices based on resources speeds up training in iid scenarios, it might hurt performance in non-iid cases, especially when resources are coupled with data distribution shifts, therefore, an environment not only enforces specific constraints on some devices but also influences the devices' data distributions. This may, for instance, be the case in an IoT system, where some devices are deployed indoors while others are outdoors, with outdoor devices being more constrained in energy and with data distribution being variant between indoor and outdoor devices. This issue is tackled in the following techniques.

Client Selection with Device Impact: Lai et al. [57] present *Oort*, a client selection framework with the aim to optimize convergence speed (w.r.t. time) by allowing for rounds with variable lengths. In difference to previous client selection techniques, Oort considers the impact a device has on the global model. Devices and round times are selected based on a utility score that includes the resources and the global model impact based on the local training loss. Li et al. [58] introduce *PyramidFL*, building on top of Oort and further improving convergence speed, mainly through a more fine-grained selection by allowing devices to train variable amounts of local epochs.

Partial Device Updates: Li et al. [61] introduce *FedProx*, which instead of dropping stragglers completely, allows for partial contributions to the global model. This is achieved by introducing an inexactness term to the devices' updates that accounts for fewer local epochs. FedProx, therefore, allows for a variable number of local steps. Especially in cases where in baseline FedAvg many devices have to be dropped, FedProx outperforms FedAvg with respect to final accuracy. Wang et al. [93] present an optimization framework to tackle stragglers in a mobile device scenario. They conduct real-world measurements on smartphone processors to measure the training throughput of various devices. Contrary to other works, they also incorporate the effects of thermal throttling into the throughput model. The framework achieves optimal utilization of devices by accounting for the devices' capabilities and splitting the devices' private data into trainable subsets. The device scheduling algorithm is also designed to be aware of non-iid data.

Efficiency Tradeoffs: Wang et al. [95] study how to effectively utilize available resources and obtain a convergence bound, highlighting how local device steps and global rounds contribute to convergence. Further, they propose an algorithm that finds a resource-efficient tradeoff between communication and computation but does not specifically cover heterogeneity over time or across devices. They show that in non-iid data scenarios, their adaptive synchronous approach outperforms asynchronous settings in terms of convergence speed. Similarly, Tran et al. [91] study the tradeoff between training time/accuracy and energy spent on training. They associate an energy cost with training throughput per time and communication bandwidth. They model energy consumption for training throughput per second and transmission of bytes per second in a wireless channel environment. A scenario with three classes of devices is studied where each device operates at certain CPU frequency levels. With that, they determine a Pareto-optimal tradeoff between time and energy in FL with heterogeneous devices. Kim and Wu present *AutoFL* [52], a system-level approach to optimize convergence speed and energy usage by training a reinforcement learning algorithm to select an optimal subset of devices for training. The server algorithm takes several device-specific factors into account, such as the number of data classes, network bandwidth, CPU, and memory contention. The solution is evaluated in a scenario with 200 mobile devices with three different device classes (high-end, mid-end, and low-end), simulating resource contention, unstable connections, and data heterogeneity.

Discussion: Presented synchronous system-level approaches optimize the round time and device selection by processing the devices' resource availability information at the server mostly on a round basis. To account for soft constraints (i.e., throughput constraints) of the devices, the server has to track and monitor the devices' state and estimated capabilities. This induces overhead on the server as well as on the devices and might be unreasonable to predict in real-world scenarios. Others, like FedProx, allow partial local training, accounting for resource changes during local rounds without server knowledge.

4.6 System-level Awareness through Asynchronous Aggregation

Another way to account for different training times is by doing weight aggregation asynchronously. Thereby, each device can download the current model at any time and, depending on its resources, upload its updated weights (visualized in Figure 3). Several papers [15, 82, 88, 108] hint that asynchronous aggregation achieves similar convergence speeds as synchronous schemes if the staleness of devices is within certain bounds. It has to be noted that most theoretical guarantees only cover asynchronous distributed SGD in convex cases. Convergence speed guarantees for asynchronous FedAvg, especially the comparison with synchronous FedAvg, is ongoing research.

Asynchronous Aggregation: Chen et al. [18] introduce **Asynchronous Online Federated Learning (ASO)** for a setting with devices under heterogeneous resources and data quantity. They use a modified version of baseline FedAvg (L_2-norm regularization on the client and weight normalization on the server) to account for the devices' data quantity and reduce the effect of the devices' models drifting away from the global model. They assume devices continuously receive a new stream of training data and do not have the memory capacity to learn in batches. The authors show that their asynchronous aggregation scheme requires less time to converge in comparison to baseline FedAvg. Similarly, *FedAsync* by Xie et al. [97] uses an asynchronous aggregation scheme with a local regularization term. The staleness problem of devices is mitigated by weighting the devices' updates contribution with a time-dependent parameter. This means that the contribution of devices that take very long, thus updating an old model state of the server, is reduced. The authors show that FedAsync outperforms FedAvg in small staleness scenarios. Sprague et al. [87] study the effects of synchronous and asynchronous FedAvg, showing that asynchronous aggregation outperforms synchronous w.r.t. convergence speed in cases where devices have different training throughput. Additionally, they study the effects of devices joining late in training, observing a disturbance in the convergence in non-iid cases. Huba et al. present *Papaya* [44], a framework for large-scale FL that supports synchronous and asynchronous aggregation. They empirically show that in large-scale (100M phones) next-word-prediction tasks, asynchronous aggregation converges faster and with higher accuracy compared to synchronous FL. In a similar experiment, synchronous FL converges slower if the aggregation waits for stragglers or reaches lower final accuracies if stragglers are discarded.

Hybrid Aggregation: Extending their previous work *TiFL* [12], Chai et al. [13] present *FedAT*, a hybrid synchronous-asynchronous approach utilizing tiers. Devices are grouped in tiers based on their performance, similarly to TiFL. While devices within one tier do synchronous aggregation, tiers asynchronously update the global model. To mitigate bias towards faster tiers (especially in non-iid cases), FedAT weights the updates of tiers, s.t. slower tiers are considered with a higher weight when updating the global model, thus allowing for equal contribution to the global model. Similarly to FedProx [61], they use a constraint term to restrict local weights to be closer to the global model. Experiments with 100 devices grouped in five tiers show that FedAT outperforms baseline FedAvg, as well as TiFL and pure asynchronous schemes like FedAsync [97].

Discussion: Presented approaches tackle soft constraints of devices by doing aggregation asynchronously. Asynchronous approaches allow for an arbitrarily high granularity in the heterogeneity, since every device can upload its model at any time. While the straggler effect can be fully addressed that way, asynchronous aggregation suffers from stale updates. The scale of heterogeneity is therefore limited by the effect of stale updates. From current research, it can be concluded that it is unknown whether synchronous or asynchronous aggregation is ultimately preferable. Results show that, depending on the assumptions, both show advantageous properties. Further theoretical work to study convergence properties beyond convex utility cases is needed.

5 OPEN PROBLEMS AND FUTURE DIRECTIONS

Current techniques covered in Section 4 have great potential to enable FL for applications with device heterogeneity. However, we identify several open problems that demand further research.

Problem 1: Maintaining effectiveness under fine-grained granularity or large-scale heterogeneity: In most of the state-of-the-art research, tackling heterogeneity focuses on accounting for soft and/or hard constraints. The attributes scale and granularity are often neglected, are hidden behind the technique, and lack discussion in the papers. The reported scale in the resources supported by the techniques ranges from 4× to 25× [12, 41, 52, 61, 71, 77, 79, 80, 85, 101] up to 100×–250× [25, 87], yet it remains unclear whether training at such high scales is still effective. Hence, while all approaches show the effectiveness of their solution in certain scenarios, it often remains unclear whether devices with low resources or stale devices can make a meaningful contribution that advances the global model. Especially in iid settings, current state-of-the-art works do not compare themselves against trivial baselines such as dropping of devices (accepting a smaller total share of data), which is the solution that current real-world FL applications like *GBoard* employ. A second trivial baseline is deploying a low-complexity model for all devices [80], which can already outperform some state-of-the-art techniques. A potential solution to maintaining effectiveness for large-scale heterogeneity with fine-grained granularity could be the interplay of system-level and NN-level approaches, as their favorable properties could complement each other. For example, an NN-architecture subset technique [25, 41] could be complemented by system-level client selection [75] or asynchronous aggregation [18, 87, 97] to increase granularity w.r.t. throughput. However, while system and NN architecture level mechanisms are often orthogonal, it remains unclear how this would affect the convergence and reachable accuracy.

Problem 2: Comparability: Current research lacks comparability w.r.t. the resource model. This is especially the case in techniques using subsets, where some model resources in terms of power usage [104], while others count the number of parameters [2, 25, 41, 79], the number of multiply accumulate-operations [41, 80], or the required training time [77]. Therefore, the supported scale of heterogeneity by the techniques is not comparable. Similarly, also the granularity of heterogeneity lacks discussion. While distillation-based approaches allow for different model architectures, only a small number of model architectures are used in the experiments (e.g., ResNet20, ResNet32 [36], and ShuffleNet [107] in FedDF [63]). The complexity differences in training these various types of networks are not further evaluated. Other approaches support two to five *groups* of devices [12, 13, 41, 91], which may lead to inefficient use of available resources, as discussed in Section 3.2. In general, different scales and granularities of the heterogeneity have to be taken into account to address real-world heterogeneity aspects such as varying peak performance and memory capacity, as well as resource contention. For a broader deployment of FL solutions in real-world use cases, these aspects require further discussion. Besides that scale of heterogeneity, also the number of devices and their share of data influences the performance of the techniques. As of now, there is no standard scenario. As a result, some approaches evaluate their techniques with over 1,000 devices, while others evaluate only a setting with two participants. Benchmark scenarios to compare FL techniques have been proposed only recently [10]. However, these benchmarks do not represent the resource constraints of devices as it is present in real-world applications. Additionally, available state-of-the-art FL simulation frameworks like *FLOWER*,[2] *FedML*,[3] *TensorFlow*

[2]https://flower.dev.
[3]https://fedml.ai.

Federated,[4] or *OpenFL*[5] do not implement device heterogeneity specifically, memory, throughput, or energy constraints. To solve this issue, first, a more device-representative benchmark for FL is required that more realistically models the environments of IoT, smartphones, or sensor networks. Second, heterogeneity support in popular FL frameworks is required.

Problem 3: Unexplored tradeoffs and non-iid data scenarios: The objectives of the discussed techniques are mostly accuracy or convergence speed. Only a few consider energy efficiency in heterogeneous settings, which is crucial in many embedded or IoT scenarios [24] where the available energy is limited. While utilizing all devices up to their capabilities speeds up training the most, it might not always be a very energy-efficient way to train the global model. Limited energy leads to a tradeoff between using the energy for communication or computation [91, 95], which has not been explored in heterogeneous FL. Besides tradeoffs between communication and computation, also tradeoffs between computation and memory exist, where intermediate results can either be stored in memory or dynamically recomputed when needed [53], which are unexplored in the context of resource-constrained FL.

Another rather unexplored problem in FL with computationally constrained devices is the effects of non-iid data. A case currently not present in the literature is the case when the data distribution is non-iid over the devices, but additionally, there is a correlation between the data and the devices' resources. Yet, a scenario like this is expected to occur in real-world FL applications [68]. A first example is a set of sensors with different power sources that sample environments that differ from each other. Similar examples can be found in a smartphone FL application. To meet a price target for certain markets, devices with different capabilities are manufactured. Different markets can lead to differences in device usage, hence, differences in the collected data. This may lead to a non-iid data scenario where weak devices hold a certain type of data that, due to fairness reasons, cannot be excluded and has to be incorporated into the global model. This kind of non-iid scenario exacerbates the need to learn from any device available.

Further research is required to identify what the effect of these correlations is and how their effects on the global model can be mitigated to enable a fair representation of any user group in the global model.

6 CONCLUSION

This survey provided an overview of FL under computation heterogeneity among the participating devices, as it occurs in many practical scenarios. We analyzed the computational constraints in smart devices that lead to heterogeneity and presented a categorization that groups the constraints into hard constraints and soft constraints that vary over devices, rounds, and time and can lead to heterogeneity of different scales with different granularities. We provided a comprehensive survey on current research on FL under heterogeneous computation constraints and how the techniques tackle the different proprieties of heterogeneity. Finally, we identify several open problems, such as the lack of comparability, problems with the solutions' effectiveness w.r.t. the heterogenities' scale and granularity, and unexplored tradeoffs.

REFERENCES

[1] Sawsan Abdulrahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. 2020. A survey on federated learning: The journey from centralized to distributed on-site learning and beyond. *IEEE IoT J.* 8, 7 (2020), 5476–5497.

[4] https://tensorflow.org/federated.
[5] https://github.com/intel/openfl.

[2] Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. 2022. FedRolex: Model-heterogeneous federated learning with rolling sub-model extraction. In *Advances in Neural Information Processing Systems*, Vol. 35. Curran Associates, Red Hook, NY, 158–171.

[3] M. M. Amiri and D. Gündüz. 2020. Federated learning over wireless fading channels. *IEEE Trans. Wireless Commun.* 19, 5 (2020), 3546–3557. https://doi.org/10.1109/TWC.2020.2974748

[4] Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. 2018. AI and Compute. Retrieved from https://openai.com/blog/ai-and-compute/.

[5] Giorgos Armeniakos, Georgios Zervakis, Dimitrios Soudris, and Jörg Henkel. 2022. Hardware approximate techniques for deep neural network accelerators: A survey. *ACM Comput. Surv.* 55, 4, Article 83 (November 2022), 36 pages. https://doi.org/10.1145/3527156

[6] Paolo Bellavista, Luca Foschini, and Alessio Mora. 2021. Decentralised learning in federated deployment environments: A system-level survey. *ACM Comput. Surv.* 54, 1 (2021), 1–38.

[7] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards federated learning at scale: System design. In *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia (Eds.), Vol. 1. 374–388.

[8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2016. Practical secure aggregation for federated learning on user-held data. arXiv:1611.04482. Retrieved from https://arxiv.org/abs/1611.04482.

[9] Sebastian Caldas, Jakub Konečny, H. Brendan McMahan, and Ameet Talwalkar. 2018. Expanding the reach of federated learning by reducing client resource requirements. arXiv:1812.07210. Retrieved from https://arixv.org/abs/1812.07210.

[10] S. Caldas, P. Wu, T. Li, J Konecnỳ, H. B. McMahan, V. Smith, and A. Talwalkar. 2019. Leaf: A benchmark for federated settings. arXiv:1812.01097. Retrieved from https://arxiv.org/abs/1812.01097.

[11] Government of California. 2020. California Consumer Privacy Act. Retrieved from https://oag.ca.gov/privacy/ccpa/regs.

[12] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. 2020. TiFL: A tier-based federated learning system. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPD'20).* Association for Computing Machinery, New York, NY, 125–136. https://doi.org/10.1145/3369583.3392686

[13] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. 2021. FedAT: A high-performance and communication-efficient federated learning system with asynchronous tiers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'21, Vol. 1).* Association for Computing Machinery, New York, NY, Article 60, 16 pages. https://doi.org/10.1145/3458817.3476211

[14] Hongyan Chang, Virat Shejwalkar, Reza Shokri, and Amir Houmansadr. 2019. Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer. arXiv:1912.11279. Retrieved from https://arxiv.org/abs/1912.11279.

[15] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2017. Revisiting distributed synchronous SGD. arXiv:1604.00981. Retrieved from https://arxiv.org/abs/1604.00981.

[16] Ling-Jyh Chen, Yao-Hua Ho, Hsin-Hung Hsieh, Shih-Ting Huang, Hu-Cheng Lee, and Sachit Mahajan. 2018. ADF: An anomaly detection framework for large-scale PM2.5 sensing systems. *IEEE IoT J.* 5, 2 (2018), 559–570. https://doi.org/10.1109/JIOT.2017.2766085

[17] Ling-Jyh Chen, Yao-Hua Ho, Hu-Cheng Lee, Hsuan-Cho Wu, Hao-Min Liu, Hsin-Hung Hsieh, Yu-Te Huang, and Shih-Chun Candice Lung. 2017. An open framework for participatory PM2.5 monitoring in smart cities. *IEEE Access* 5 (2017), 14441–14454. https://ieeexplore.ieee.org/document/7970115.

[18] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala. 2020. Asynchronous online federated learning for edge devices with non-IID data. In *Proceedings of the 8th Conference IEEE International Conference on Big Data (Big Data'20).* IEEE, 15–24. https://doi.org/10.1109/BigData50022.2020.9378161

[19] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. 2020. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intell. Syst.* 35, 4 (2020), 83–93.

[20] Ayush Chopra, Surya Kant Sahu, Abhishek Singh, Abhinav Java, Praneeth Vepakomma, Vivek Sharma, and Ramesh Raskar. 2021. AdaSplit: Adaptive trade-offs for resource-constrained distributed deep learning. arXiv:2112.01637. Retrieved from https://arxiv.org/abs/2112.01637.

[21] Bekir Sait Ciftler, Abdullatif Albaseer, Noureddine Lasla, and Mohamed Abdallah. 2020. Federated learning for localization: A privacy-preserving crowdsourcing method. online. arXiv:2001.01911. Retrieved from https://arxiv.org/abs/2001.01911.

[22] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. 2017. EMNIST: An extension of MNIST to handwritten letters. arXiv:1702.05373. Retrieved from https://arxiv.org/abs/1702.05373.

[23] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Sign. Process. Mag.* 29, 6 (2012), 141–142.

[24] Sauptik Dhar, Junyao Guo, Jiayi (Jason) Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. 2021. A survey of on-device machine learning: An algorithms and learning theory perspective. *ACM Trans. Internet Things* 2, 3, Article 15 (July 2021), 49 pages. https://doi.org/10.1145/3450494

[25] Enmao Diao, Jie Ding, and Vahid Tarokh. 2020. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. In *Proceedings of the 8th Conference International Conference on Learning Representations (ICLR'20)*, Vol. 1.

[26] Matthieu Dubarry, Vojtech Svoboda, Ruey Hwu, and Bor Yann Liaw. 2007. Capacity and power fading mechanism identification from a commercial cell evaluation. *J. Power Sourc.* 165, 2 (2007), 566–572.

[27] Lukasz Dudziak, Stefanos Laskaridis, and Javier Fernandez-Marques. 2022. FedorAS: Federated Architecture Search under system heterogeneity. arXiv:2206.11239. Retrieved from https://arxiv.org/abs/2206.11239.

[28] Ahmed Roushdy Elkordy and A. Salman Avestimehr. 2020. Secure aggregation with heterogeneous quantization in federated learning. arXiv:2009.14388. Retrieved from https://arxiv.org/abs/2009.14388.

[29] Ozan Erdinc, Bulent Vural, and Mehmet Uzunoglu. 2009. A dynamic lithium-ion battery model considering the effects of temperature and capacity fading. In *Proceedings of the 2nd Conference International Conference on Clean Electrical Power*. IEEE, 383–386.

[30] EU. 2020. European Union's General Data Protection Regulation (GDPR). Retrieved from https://gdpr.eu/.

[31] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom'18)*. Association for Computing Machinery, New York, NY, 115–127. https://doi.org/10.1145/3241539.3241559

[32] Dashan Gao, Ce Ju, Xiguang Wei, Yang Liu, Tianjian Chen, and Qiang Yang. 2019. Hhhfl: Hierarchical heterogeneous horizontal federated learning for electroencephalography. arXiv:1909.05784. Retrieved from https://arxiv.org/abs/1909.05784.

[33] N. Garg and R. Garg. 2017. Energy harvesting in IoT devices: A survey. In *Proceedings of the International Conference on Intelligent Sustainable Systems (ICISS'17)*. IEEE, 127–131. https://doi.org/10.1109/ISS1.2017.8389371

[34] Chaoyang He, Murali Annavaram, and Salman Avestimehr. 2020. Group knowledge transfer: Federated learning of large CNNs at the edge. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Red Hook, NY, 14068–14080.

[35] Chaoyang He, Erum Mushtaq, Jie Ding, and Salman Avestimehr. 2022. FedNAS: Federated Deep Learning via Neural Architecture Search. Retrieved from https://openreview.net/forum?id=1OHZX4YDqhT.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. arXiv:1512.03385. Retrieved from https://arxiv.org/abs/1512.03385.

[37] Jörg Henkel, Heba Khdr, Santiago Pagani, and Muhammad Shafique. 2015. New trends in dark silicon. In *Proceedings of the Design Automation Conference (DAC'15)*, Vol. 52. IEEE and Association for Computing Machinery, 1–6.

[38] Jörg Henkel, Heba Khdr, and Martin Rapp. 2019. Smart thermal management for heterogeneous multicores. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'19)*. IEEE, 132–137.

[39] Chan Yun Hin and Ngai Edith. 2021. FedHe: Heterogeneous models and communication-efficient federated learning. arXiv:2110.09910. Retrieved from https://arxiv.org/abs/2110.09910.

[40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. arXiv:1503.02531. Retrieved from https://arxiv.org/abs/1503.02531.

[41] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. 2021. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. In *Advances in Neural Information Processing Systems*, Vol. 34. 1–12.

[42] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861. Retrieved from https://arxiv.org/abs/1704.04861.

[43] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861. Retrieved from https://arxiv.org/abs/1704.04861.

[44] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, et al. 2022. Papaya: Practical, private, and scalable federated learning. *Proc. Mach. Learn. Syst.* 4 (2022), 814–832.

[45] Ahmed Imteaj and M. Hadi Amini. 2021. FedAR: Activity and resource-aware federated learning model for distributed mobile robots. arXiv:2101.03705. Retrieved from https://arxiv.org/abs/2101.03705.

[46] Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M. Hadi Amini. 2020. Federated learning for resource-constrained IoT devices: Panoramas and state-of-the-art. arXiv:2002.10610. Retrieved from https://arxiv.org/abs/2002.10610.

[47] Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M. Hadi Amini. 2021. A survey on federated learning for resource-constrained IoT devices. *IEEE IoT J.* 9, 1 (2021), 1–24.

[48] Yuang Jiang, Shiqiang Wang, Bong Jun Ko, Wei-Han Lee, and Leandros Tassiulas. 2019. Model pruning enables efficient federated learning on edge devices. arXiv:1909.12326. Retrieved from https://arxiv.org/abs/1909.12326.

[49] Ce Ju, Dashan Gao, Ravikiran Mane, Ben Tan, Yang Liu, and Cuntai Guan. 2020. Federated transfer learning for EEG signal classification. In *Proceedings of the 42nd Conference Engineering in Medicine & Biology Society (EMBC'20).* IEEE, 3040–3045.

[50] P. Kairouz, H. McMahan, B. Avent, Aurélien Bellet, Mehdi Bennis, A. Bhagoji, Keith Bonawitz, Z. Charles, Graham Cormode, R. Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, D. Evans, Josh Gardner, Zachary A. Garrett, A. Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Z. Harchaoui, Chaoyang He, Lie He, Z. Huo, B. Hutchinson, Justin Hsu, M. Jaggi, T. Javidi, Gauri Joshi, M. Khodak, Jakub Konecný, A. Korolova, F. Koushanfar, O. Koyejo, T. Lepoint, Yang Liu, P. Mittal, M. Mohri, R. Nock, Ayfer üzgür, R. Pagh, Mariana Raykova, Hang Qi, D. Ramage, R. Raskar, D. Song, Weikang Song, S. Stich, Ziteng Sun, A. T. Suresh, Florian Tramér, Praneeth Vepakomma, Jianyu Wang, L. Xiong, Zheng Xu, Q. Yang, F. Yu, Han Yu, and Sen Zhao. 2019. Advances and open problems in federated learning. arXiv:1912.04977. Retrieved from https://arxiv.org/abs/1912.04977.

[51] Latif U. Khan, Walid Saad, Zhu Han, Ekram Hossain, and Choong Seon Hong. 2021. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Commun. Surv. Tutor.* 32, 3 (2021), 1759–1799.

[52] Young Geun Kim and Carole-Jean Wu. 2021. AutoFL: Enabling heterogeneity-aware energy efficient federated learning. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'21).* Association for Computing Machinery, New York, NY, 183–198. https://doi.org/10.1145/3466752.3480129

[53] Marisa Kirisame, Steven Lyubomirsky, Altan Haan, Jennifer Brennan, Mike He, Jared Roesch, Tianqi Chen, and Zachary Tatlock. 2020. Dynamic tensor rematerialization. arXiv:2006.09616. Retrieved from https://arxiv.org/abs/2006.19616.

[54] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. arXiv:1610.05492. Retrieved from https://arxiv.org/abs/1610.05492.

[55] Alex Krizhevsky, Geoffrey Hinton, and others. 2009. Learning multiple layers of features from tiny images. Technical Report.

[56] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. 2020. Survey of personalization techniques for federated learning. In *Proceedings of the 4th World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4'20).* IEEE, 794–797. https://doi.org/10.1109/WorldS450073.2020.9210355

[57] Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient federated learning via guided participant selection. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI'21).* USENIX Association, Virtual, 19–35.

[58] Chenning Li, Xiao Zeng, Mi Zhang, and Zhichao Cao. 2022. PyramidFL: A fine-grained client selection framework for efficient federated learning. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. Association for Computing Machinery, New York, NY, 158–171. https://doi.org/10.1145/3495243.3517017

[59] Daliang Li and Junpu Wang. 2019. FedMD: Heterogenous federated learning via model distillation. In *Proceedings of the Workshop on Federated Learning for Data Privacy and Confidentiality*, Vol. 33. 1–4.

[60] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Sign. Process. Mag.* 37, 3 (2020), 50–60. https://doi.org/10.1109/MSP.2020.2975749

[61] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. In *Proceedings of Machine Learning and Systems*, Vol. 2. MLSys, Santa Clara, CA, 429–450.

[62] Xinle Liang, Yang Liu, Tianjian Chen, Ming Liu, and Qiang Yang. 2022. Federated transfer reinforcement learning for autonomous driving. *Federated and Transfer Learning*. Springer, 357–371.

[63] Tao Lin, Lingjing Kong, Sebastian U. Stich, and Martin Jaggi. 2020. Ensemble distillation for robust model fusion in federated learning. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Red Hook, NY, 2351–2363.

[64] Boyi Liu, Lujia Wang, and Ming Liu. 2019. Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems. *IEEE Robot. Autom. Lett.* 4, 4 (2019), 4555–4562.

[65] Boyi Liu, Lujia Wang, Ming Liu, and Cheng-Zhong Xu. 2019. Federated imitation learning: A privacy considered imitation learning framework for cloud robotic systems with heterogeneous sensor data. arxiv:cs.RO. 1909.00895.

[66] Detian Liu and Yang Cao. 2021. Federated neural architecture search evolution and open problems: An overview. In *Proceedings of the International Conference on Bio-Inspired Computing: Theories and Applications*, Vol. 16. Springer, 330–345.

[67] Sin Kit Lo, Qinghua Lu, Chen Wang, Hye-Young Paik, and Liming Zhu. 2021. A systematic literature review on federated machine learning: From a software engineering perspective. *ACM Comput. Surv.* 54, 5 (2021), 1–39.

[68] Kiwan Maeng, Haiyu Lu, Luca Melis, John Nguyen, Mike Rabbat, and Carole-Jean Wu. 2022. Towards fair federated recommendation learning: Characterizing the inter-dependence of system and data heterogeneity. arXiv:2206.02633. Retrieved from https://arxiv.org/abs/2206.02633.

[69] James Manyika, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, and Dan Aharon. 2015. *Unlocking the Potential of the Internet of Things.* McKinsey Global Institute.

[70] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics.* PMLR, 1273–1282.

[71] Yiqun Mei, Pengfei Guo, Mo Zhou, and Vishal Patel. 2022. Resource-adaptive federated learning with all-in-one neural composition. In *Advances in Neural Information Processing Systems*, Vol. 35. Curran Associates, Red Hook, NY, 1–10.

[72] J. Mills, J. Hu, and G. Min. 2020. Communication-efficient federated learning for wireless edge intelligence in IoT. *IEEE IoT J.* 7, 7 (2020), 5986–5994. https://doi.org/10.1109/JIOT.2019.2956615

[73] Xiaopeng Mo and Jie Xu. 2021. Energy-efficient federated edge learning with joint communication and computation design. *J. Commun. Inf. Netw.* 6, 2 (2021), 110–124. https://doi.org/10.23919/JCIN.2021.9475121

[74] Erum Mushtaq, Chaoyang He, Jie Ding, and Salman Avestimehr. 2021. SPIDER: Searching Personalized Neural Architecture for Federated Learning. arXiv:2112.13939. Retrieved from https://arxiv.org/abs/2112.13939

[75] T. Nishio and R. Yonetani. 2019. Client selection for federated learning with heterogeneous resources in mobile edge. In *Proceedings of the IEEE International Conference on Communications (ICC'19)*, Vol. 53. IEEE, 1–7.

[76] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Vol. 32. Curran Associates, Red Hook, NY, 8024–8035.

[77] Kilian Pfeiffer, Martin Rapp, Ramin Khalili, and Jörg Henkel. 2022. CoCo-FL: Communication- and computation-aware federated learning via partial NN freezing and quantization. arXiv:2203.05468. Retrieved from https://arxiv.org/abs/2203.05468.

[78] Jason Posner, Lewis Tseng, Moayad Aloqaily, and Yaser Jararweh. 2021. Federated learning in vehicular networks: Opportunities and solutions. *IEEE Netw.* 35 (2021), 1–12.

[79] Xinchi Qiu, Javier Fernandez-Marques, Pedro P. B. Gusmao, Yan Gao, Titouan Parcollet, and Nicholas Donald Lane. 2022. ZeroFL: Efficient on-device training for federated learning with local sparsity. In *Proceedings of the International Conference on Learning Representations (ICLR'22)*, Vol. 9. OpenReview, 1–10.

[80] Martin Rapp, Ramin Khalili, Kilian Pfeiffer, and Jörg Henkel. 2022. DISTREAL: Distributed resource-aware learning in heterogeneous systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. AAAI, 1–12.

[81] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. 2020. FedPAQ: A communication-efficient federated learning method with periodic averaging and quantization. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 108)*, Silvia Chiappa and Roberto Calandra (Eds.). PMLR, 2021–2031.

[82] Amirhossein Reisizadeh, Isidoros Tziotis, Hamed Hassani, Aryan Mokhtari, and Ramtin Pedarsani. 2020. Straggler-resilient federated learning: Leveraging the interplay between statistical accuracy and system heterogeneity. arXiv:2012.14453. Retrieved from https://arxiv.org/abs/2012.14453.

[83] Farzad Samie, Lars Bauer, and Jörg Henkel. 2019. From cloud down to things: An overview of machine learning in internet of things. *IEEE IoT J.* 6, 3 (2019), 4921–4934.

[84] Yuris Mulya Saputra, Dinh Thai Hoang, Diep N. Nguyen, Eryk Dutkiewicz, Markus Dominik Mueck, and Srikathyayani Srikanteswara. 2019. Energy demand prediction with federated learning for electric vehicle networks. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM'19).* IEEE, 1–6.

[85] Tao Shen, Jie Zhang, Xinkang Jia, Fengda Zhang, Gang Huang, Pan Zhou, Kun Kuang, Fei Wu, and Chao Wu. 2020. Federated mutual learning. arXiv:2006.16765. Retrieved from https://arxiv.org/abs/2006.16765.

[86] Yuanming Shi, Kai Yang, Tao Jiang, Jun Zhang, and Khaled B. Letaief. 2020. Communication-efficient edge AI: Algorithms and systems. *IEEE Commun. Surv. Tutor.* 22, 4 (2020), 2167–2191. https://doi.org/10.1109/COMST.2020.3007787

[87] Michael R. Sprague, Amir Jalalirad, and Marco Scavuzzo. 2018. Asynchronous federated learning for geospatial applications. In *Proceedings of the 29th Conference European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD'18) Workshops*. Springer International, 21–28.

[88] Sebastian U. Stich. 2019. Local SGD converges fast and communicates little. In *Proceedings of the International Conference on Learning Representations*, Vol. 7. ICLR, 1–12.

[89] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, Vol. 36. ICML, 6105–6114.

[90] Yue Tan, Guodong Long, Lu Liu, Tianyi Zhou, Qinghua Lu, Jing Jiang, and Chengqi Zhang. 2022. Fedproto: Federated prototype learning over heterogeneous devices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. AAAI, 1–12.

[91] Nguyen H. Tran, Wei Bao, Albert Zomaya, Minh N. H. Nguyen, and Choong Seon Hong. 2019. Federated learning over wireless networks: Optimization model design and analysis. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'19)*, Vol. 38. IEEE, 1387–1395.

[92] Victor M. Van Santen, Javier Martin-Martinez, Hussam Amrouch, Montserrat Maqueda Nafria, and Jörg Henkel. 2017. Reliability in super-and near-threshold computing: A unified model of RTN, BTI, and PV. *IEEE Trans. Circ. Syst. I: Regul. Pap.* 65, 1 (2017), 293–306.

[93] Cong Wang, Yuanyuan Yang, and Pengzhan Zhou. 2020. Towards efficient scheduling of federated mobile devices under computational and statistical heterogeneity. *IEEE Trans. Parallel Distrib. Syst.* 32, 2 (2020), 394–410.

[94] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. 2020. Federated learning with matched averaging. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*, Vol. 8. ICLR, 1–12.

[95] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan. 2019. Adaptive federated learning in resource constrained edge computing systems. *IEEE J. Select. Areas Commun.* 37, 6 (2019), 1205–1221. https://doi.org/10.1109/JSAC.2019.2904348

[96] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. 2020. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Trans. Inf. Forens. Secur.* 15 (2020), 3454–3469.

[97] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2020. Asynchronous federated optimization. arXiv:1903.03934. Retrieved from https://arxiv.org/abs/1903.03934.

[98] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. 2021. Asynchronous federated learning on heterogeneous devices: A survey. arXiv:2109.04269. Retrieved from https://arxiv.org/abs/2109.04269.

[99] Zirui Xu, Zhao Yang, Jinjun Xiong, Jianlei Yang, and Xiang Chen. 2019. Elfish: Resource-aware federated learning on heterogeneous edge devices. arXiv:1912.01684. Retrieved from https://arxiv.org/abs/1912.01684.

[100] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving Google keyboard query suggestions. arXiv:1812.02903. Retrieved from https://arxiv.org/abs/1812.02903.

[101] Dezhong Yao, Wanning Pan, Yao Wan, Hai Jin, and Lichao Sun. 2021. FedHM: Efficient federated learning for heterogeneous models via low-rank factorization. arXiv:2111.14655. Retrieved from https://arxiv.org/abs/2111.14655.

[102] Dixi Yao, Lingdong Wang, Jiayu Xu, Liyao Xiang, Shuo Shao, Yingqi Chen, and Yanjun Tong. 2021. Federated model search via reinforcement learning. In *Proceedings of the IEEE 41st International Conference on Distributed Computing Systems (ICDCS'21).* IEEE, Curran Associates, Red Hook, NY, 830–840.

[103] Xuefei Yin, Yanming Zhu, and Jiankun Hu. 2021. A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions. *ACM Comput. Surv.* 54, 6 (2021), 1–36.

[104] R. Yu and P. Li. 2021. Toward resource-efficient federated learning in mobile edge computing. *IEEE Netw.* 35, 1 (2021), 148–155. https://doi.org/10.1109/MNET.011.2000295

[105] Binhang Yuan, Song Ge, and Wenhui Xing. 2020. A federated learning framework for healthcare IoT devices. arXiv:2005.05083. Retrieved from https://arxiv.org/abs/2005.05083.

[106] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. 2019. Bayesian nonparametric federated learning of neural networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19),* Proceedings of Machine Learning Research, Vol. 97, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). ICML, 7252–7261.

[107] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the 27th Conference IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 6848–6856.

[108] Zhengyuan Zhou, Panayotis Mertikopoulos, Nicholas Bambos, Peter Glynn, Yinyu Ye, Li-Jia Li, and Li Fei-Fei. 2018. Distributed asynchronous optimization with unbounded delays: How slow can you go? In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80. ICML, 5970–5979.

Received 6 July 2022; revised 1 February 2023; accepted 2 May 2023