# Network-Aware Optimization of Distributed Learning for Fog Computing

Yuwei Tu, Yichen Ruan, Su Wang, Satyavrat Wagle,
Christopher G. Brinton, *Senior Member, IEEE,* and Carlee Joe-Wong, *Member, IEEE*

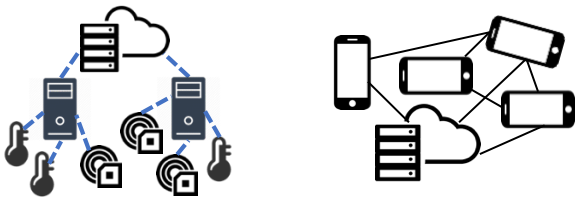
***Abstract*—Fog computing promises to enable machine learning tasks to scale to large amounts of data by distributing processing across connected devices. Two key challenges to achieving this goal are (i) heterogeneity in devices' compute resources and (ii) topology constraints on which devices can communicate with each other. We address these challenges by developing the first network-aware distributed learning optimization methodology where devices optimally share local data processing and send their learnt parameters to a server for aggregation at certain time intervals. Unlike traditional federated learning frameworks, our method enables devices to offload their data processing tasks to each other, with these decisions determined through a convex data transfer optimization problem that trades off costs associated with devices processing, offloading, and discarding data points. We analytically characterize the optimal data transfer solution for different fog network topologies, showing for example that the value of offloading is approximately linear in the range of computing costs in the network. Our subsequent experiments on testbed datasets we collect confirm that our algorithms are able to improve network resource utilization substantially without sacrificing the accuracy of the learned model. In these experiments, we also study the effect of network dynamics, quantifying the impact of nodes entering/exiting the network on model learning and resource costs.**

***Index Terms*—federated learning, offloading, fog computing**

## I. Introduction

New technologies like autonomous cars and smart factories are coming to rely extensively on data-driven machine learning (ML) algorithms [2]–[4] to produce near real-time insights based on historical data. Training ML models at realistic scales, however, is challenging, given the enormous computing power required to process today's data volumes. The collected data is also dispersed across networks of devices, while ML models are traditionally managed in a centralized manner [5].

Fortunately, the rise in data generation in networks has been accompanied by a corresponding rise in the computing power of networked devices. Thus, a possible solution for training and making real-time inferences from data-driven ML algorithms lies in the emerging paradigm of fog computing, which aims to design systems, architectures, and algorithms that leverage an aggregation of device capacities between the network edge and cloud [6]. Deployment of 5G wireless networks and the Internet of Things (IoT) is accelerating adoption of this computing paradigm by expanding the set of connected devices with compute capabilities and enabling direct device-to-device communications [7]. Though centralized ML algorithms are not optimized for such environments, distributed data analytics is expected to be a major driver of 5G adoption [8].



(a) Hierarchical (b) Social network

Fig. 1: Cartoon illustrations of two example topologies for fog computing that we consider. In the hierarchical case, less powerful devices are connected to more powerful ones, while for the social network, connections are denser and devices tend to be similar.

Initial efforts in decentralizing ML have focused on decomposing model parameter updates over several nodes, typically managed by a centralized serving entity [9], [10]. Most of these methods, however, implicitly assume idealized network topologies where node and link properties are homogeneous. Fog environments, by contrast, are characterized by devices' heterogeneity both in available compute resources and in connectivity with each other, e.g., due to power constraints, mobility, or privacy considerations. For example. consider the two common fog topologies depicted in Figure 1. In the hierarchical scenario, less powerful edge devices are connected to more powerful nodes like edge servers. In the social network case, devices tend to have similar compute resources, but connectivity may vary significantly depending on levels of trust between users [11].

A central question that arises, then, in adapting ML methodologies to fog environments is: *How should each fog device contribute to the ML training and inference?* We answer this question by developing a methodology for *optimizing the distribution of processing across a network of fog devices*.

### A. Machine Learning in Fog Environments

ML models are generally trained by iteratively updating estimates of parameter values, such as weights in a neural network, that best "fit" empirical data through data processing at a node or set of devices. We face two major challenges in adapting such training to fog networking environments: *(i) het-*

This work was presented in part at the 2020 IEEE Conference on Computer Communications (INFOCOM) [1].

S. Wang and C. Brinton are with the School of Electrical and Computer Engineering at Purdue University. email: {wang2506,cgb}@purdue.edu

Y. Ruan, S. Wagle, and C. Joe-Wong are with the Department of Electrical and Computer Engineering at Carnegie Mellon University. email: {yichenr, srwagle, cjoewong}@andrew.cmu.edu

*erogeneity in devices' compute resources* and *(ii) constraints on devices' abilities to communicate with each other*. We outline these characteristics, and the potential benefits enabled by our network-aware distributed learning methodology, in some key applications below:

**Privacy-sensitive applications.** Many ML applications learn models on sensitive user data, e.g., for health monitoring [4]. Due to privacy concerns, most of these applications have devices train their models on local data to avoid revealing data to untrustworthy nodes [11]. Offloading ML data processing to trusted devices, e.g., those owned by friends, however, can reduce training times and improve model accuracy.

**Internet-connected vehicles** can collaboratively learn about their environment [2], e.g., by combining their data with that of road sensors to infer current road or traffic conditions. Since sensors have less computing capabilities than vehicles, they will likely offload their data to vehicles or roadside units for processing. This offloading must adapt as vehicles move and their connectivity with (stationary) sensors changes.

**Augmented reality (AR)** uses ML algorithms for e.g., image recognition [3] to overlay digital content onto users' views of an environment. A network of AR-enabled devices can distributedly train ML models, but may exhibit significant heterogeneity: they can range from generic smartphones to AR-specific headsets, with different battery levels. As the users move, connectivity between devices will also change.

**Industrial IoT.** 5G networks will allow sensors that power control loops in factory production lines to communicate across the factory floor [2], [12], enabling distributed ML algorithms to use this data for, e.g., predicting production delays. It is an open question to determine which controllers should process data from which sensors: this depends on sensor-controller connectivities, which may vary with factory activity.

### B. Outline and Summary of Contributions

First, Section II differentiates our work from relevant literature. To the best of our knowledge, we are the first to optimize the distribution of ML data processing (i.e., training) tasks across fog nodes, leading to several contributions:

**Formulating the task distribution problem** (Section III). In deciding which devices should process which datapoints, our formulation accounts for resource limitations and model accuracy. While ideally more of the data would be processed at devices with more computing resources, sending data samples to such devices may overburden the network. Moreover, processing too many data samples can incur large processing costs relative to the gain in model accuracy. We derive new bounds (Theorem 1) on the model accuracy when data can be moved between devices, and show that the optimal task distribution problem can be formulated as a convex optimization that can be solved rapidly even for large networks.

**Characterizing the optimal task distribution** (Section IV). Solving the optimization problem formulated in Section III requires specifying network characteristics that may not be known in advance, e.g., the number of datapoints that each device can process in a single timeslot. We analyze the expected deviations from our assumptions in Section III to derive guidelines on how these characteristics should be set (Theorem 2). We then derive the optimal task distributions for typical fog network topologies (Theorems 3 and 4) and use them to estimate the value (i.e., reduction in processing costs) of allowing devices to move processing tasks to other devices (Theorems 5 and 6).

**Experimental validation** (Section V). We train classification models on the MNIST dataset to validate our algorithms. We use data traces from a Raspberry Pi testbed to emulate network delays and compute resource availability. Our proposed algorithm nearly halves the processing overhead yet achieves an accuracy comparable to centralized model training.

We discuss potential extensions of our work and conclude in Section VI.

## II. Related Work

We contextualize our work within prior results on (i) federated learning algorithms and (ii) methods for offloading ML tasks from mobile devices to edge servers.

### A. Federated Learning

In classical distributed learning, multiple "workers" each compute a gradient or parameter value on their own local data. These results are aggregated at a central server, and updated parameter values are sent back to the workers to begin another round of local computations. In the federated learning framework, devices instead perform a *series* of local updates between aggregations, and send their resulting model parameters to the server [10], [13], [14]. This framework preserves user privacy by keeping data at local devices [15] and reduces the amount of communication between devices and the central server.

Since its proposition in [10], federated learning has generated significant research interest; see [16] for a comprehensive survey. Compared to traditional distributed learning, federated learning introduces two new challenges: firstly, data may not be identically and independently distributed as devices locally generate the samples they process; and secondly, in fog/edge networks, devices may have limited ability to carry out and communicate local updates due to resource constraints. Many works have attempted to address the first challenge; for instance, [17] showed that sharing small subsets of user data can yield significant gains in model accuracy, and [18] trains user-specific models under a multi-task learning framework to fuse these individual models together. When the devices attempt to learn a single model, recent efforts have considered optimizing the frequency of parameter aggregations according to a fixed budget of network and computing resources [5], or adopting a peer-to-peer framework in which parameter updates are shared with neighboring devices instead of a central server [19].

Existing research has also sought to address the second challenge, particularly in minimizing the communication costs in federated scenarios. [20] proposes methods to reduce uplink costs by restricting the parameter space and compressing model updates prior to transmission. [21] proposes a method

for thresholding updates for transmission, while the method in [22] only communicates the most important individual gradient results. [23] broadens the results in [22] such that both downlink and uplink are compressed, [24] aggregates subsets of the network, and [25] only utilizes aggregations between one-hop neighbors. For wireless networks in particular, [26] proposes methods to minimize power consumption and training time among devices in federated learning.

Different from these works, in this paper, we focus on optimally distributing parameter computations between devices, and simultaneously optimizing the compute-communication tradeoffs inherent in fog scenarios.

### B. Offloading

Fog computing introduces opportunities to pool network resources and maximize the use of idle computational/storage power in completing resource-intensive activities [6]. Offloading mechanisms can improve system performance when there are high-bandwidth connections available to alleviate the load on constrained mobile devices. Offloading has been seen to significantly accelerate machine learning tasks such as linear regression training [27] and neural network inference [28]. Existing literature has also considered splitting the inference of different layers in deep neural networks between fog devices and edge/cloud servers to improve convergence speed. Specifically, [29] proposed two network-dependent schemes and a min-cut problem to accelerate the training phase, while [30] developed an architecture wherein edge devices rely on local inferences when reliable and transmit uncertain classification results to the cloud, to take advantage of the server's superior computational capabilities.

In general, existing works on deep learning in fog/edge computing have focused on network factors, such as cost/latency, rather than model factors, such as loss [33]. For example, [31] and [32] develop offloading schemes to maximize throughput in wireless and sensor IoT networks, respectively. Our methodology considers more general machine learning models, optimizes tradeoffs between network cost and model accuracy, and provides theoretical performance bounds not found in prior works.

## III. Model and Optimization Formulation

In this section, we define our model for fog networks (Section III-A) and machine learning training (Section III-B), and then formulate the task distribution optimization problem (Section III-C).

### A. Fog Computing System Model

We consider a set $V$ of $n$ fog devices forming a network, an aggregation server $s$, and discrete time intervals $t = 1, \dots, T$ as the period for training an ML model. Each device, e.g., a sensor or smartphone, can both collect data and process it to contribute to the ML task. The server $s$ aggregates the results of each device's local analysis, as will be explained in Section III-B. Both the length and number of time intervals may depend on the specific ML application. In each interval $t$, we suppose a subset of devices $V(t)$, indexed by $i$, is active (i.e., available to collect and/or process data). For simplicity of notation, we omit $i$'s dependence on $t$.

*1) Data collection and processing:* We use $D_i(t)$ to denote the set of data collected by device $i \in V(t)$ for the ML task at time $t$; $d \in D_i(t)$ denotes each datapoint, $d$. Note, $D_i(t) = \emptyset$ if a device does not collect data at time $t$. $G_i(t)$, by contrast, denotes the set of datapoints *processed* by each device at time $t$, for the ML task; our optimization problem in Section III-C relates the $G_i(t)$ to the datasets $D_i(t)$. In conventional distributed learning frameworks, $D_i(t) = G_i(t)$, as all devices process the data they collect [5]; separating these variables is one of our main contributions. We suppose that each device $i$ can process up to $C_i(t)$ datapoints at each time $t$, where $c_i(t)$ represents the average cost per datapoint of processing $C_i(t)$ datapoints at device $i$ and at time $t$. These might, for example, represent the battery level; devices with low battery will have lower capacities $C_i(t)$ and higher costs $c_i(t)$.

*2) Fog network connectivity:* The devices $V$ are connected to each other via a set $E$ of directed links, with $(i, j) \in E$ denoting a link from device $i$ to $j$, and $E(t) \subseteq E$ denoting the set of functioning links at time $t$. The overall system then can be described as a directed graph $(\{s, V\}, E)$ with vertices $V$ representing the devices and edges $E$ the links between them. We suppose that $(\{s, V(t)\}, E(t))$ is fully connected at each time $t$ and that links between devices are single-hop, i.e., devices do not use each other as relays except possibly to communicate with the server. Note that the scenarios outlined in Section I-A each can be modeled with this architecture: in smart factories, for example, a subset of the floor sensors (i.e., fog devices) connect to each controller (i.e., edge server). Each link $(i, j) \in E(t)$ is characterized by a capacity $C_{ij}(t)$, determined by the bandwidth of the link between devices $i$ and $j$ (i.e., the maximum datapoints they can transfer per unit time), and a "cost of connectivity" $c_{ij}(t)$. This cost may reflect network conditions (e.g., signal strengths, congestion) or a desire for privacy, and will be high if sending from $i$ to $j$ is less desirable at $t$.

*3) Data structure:* The contents of each datapoint $d$ take the form $(x_d, y_d)$, where $x_d$ is an attribute/feature vector and $y_d$ is an associated label for model learning [1]. We use $D_V = \cup_{i,t} D_i(t)$ to denote the full set of datapoints collected by all devices over all time. For simplicity, we follow prior work [34], [35] and model the data collection at device $i$ as points being selected uniformly at random from a (usually unknown) distribution $\mathcal{D}_i$. In practice, the $\mathcal{D}_i$ can evolve over time, but we assume this evolution is slow compared to the time horizon $T$. We use $\mathcal{D} = \cup_i \mathcal{D}_i$ to denote the global distribution induced by these $\mathcal{D}_i$. Note this assumption implies the relationship between $x_d$ and $y_d$ is temporally invariant, which is common in the applications discussed in Section I-A, e.g., image recognition from road cameras at fixed locations or AR users with random mobility patterns. We will use such a dataset for evaluation in Section V.

[1] For unsupervisd ML tasks, there is no $y_d$

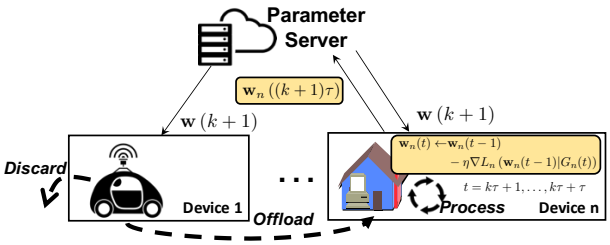



Fig. 2: Federated learning updates between aggregations $k$ and $k+1$, in our system. Device 1 discards all of its data or offloads it to device $n$, which computes $\tau$ gradient updates on its local data. The final parameter values are averaged at the parameter server, with the result sent back to the devices to begin a new iteration.

### B. Machine Learning Model

Our goal is to learn a parameterized model that outputs $y_d$ given the input feature vector $x_d$. We use the vector $w$ to denote the set of model parameters, whose values are chosen so as to minimize a loss function $L(w|\mathcal{D})$ that is defined for the specific ML model (e.g., squared error for linear regression, cross-entropy loss for multi-class classifiers [36]). Since the overall distributions $\mathcal{D}_i$ are unknown, instead of minimizing $L(w|\mathcal{D})$ we minimize the empirical loss function, as commonly done:

$$\underset{w}{\text{minimize}} \;\; L(w|D_V) = \frac{\sum_{t=1}^{T}\sum_{i\in V(t)}\sum_{d\in G_i(t)} l(w, x_d, y_d)}{|D_V|}, \tag{1}$$

where $l(w, x_d, y_d)$ is the error for datapoint $d$, and $|D_V|$ is the number of datapoints. Note that the function $l$ may include regularization terms that aim to prevent model overfitting [10].

Fog computing allows (1) to be solved in a distributed manner through the network: instead of computing the solution at the server $s$, we can leverage computations at each device $i$, so long as local resources are available. Below, we follow the commonly used federated averaging framework [5] in specifying these local computations and the subsequent global aggregation by the server in each iteration, illustrated by device $n$ in Figure 2. To avoid excessive re-optimization at each device, we suppose that they execute the same local updating algorithm regardless of $G_i(t)$. We adjust the server averaging to account for the amount of data each device processes.

*1) Local loss minimization:* In order to solve (1) in a distributed manner, we first decompose the empirical loss function into a weighted sum of local loss functions

$$L_i(w_i|G_i) = \frac{\sum_{t=1}^{T}\sum_{d\in G_i(t)} l(w, x_d, y_d)}{|G_i|}, \tag{2}$$

where $G_i \equiv \cup_{t\leq T} G_i(t)$ denotes the set of datapoints processed by device $i$ over all times. The global loss in (1) is then equal to $L(w|D_V) = \sum_i L_i(w|G_i)\,|G_i|\,/\,|D_V|$ if $\cup_i G_i = D_V$, i.e., if all datapoints $d \in D_V$ are eventually processed at some device.

Due to the inherent complexity of most ML models, loss functions such as (2) are typically minimized using gradient descent techniques [10]. Specifically, the devices update their local parameter estimates at $t$ according to

$$w_i(t) = w_i(t-1) - \eta(t)\nabla L_i(w_i(t-1)|G_i(t)), \tag{3}$$

where $\eta(t) > 0$ is the step size, which often decreases with $t$, and $\nabla L_i(w_i(t-1)|G_i(t)) = \sum_{d\in G_i(t)} \nabla l(w_i(t-1), x_d, y_d)/|G_i(t)|$ is the gradient with respect to $w$ of the average loss of points in the current dataset $G_i(t)$ at the parameter value $w_i(t-1)$. We define the loss only on the current dataset $G_i(t)$ since future data in $G_i$ has not yet been revealed; since we assume each node's data is i.i.d. over time, we can view $L_i(w_i(t-1)|G_i(t))$ as approximating the local loss $L_i(w_i|G_i)$. One can then interpret the computational cost $c_i(t)$ of processing datapoint $d$ as the cost of computing the gradient $\nabla l(w_i(t-1), x_d, y_d)$. If the local data distributions $\mathcal{D}_i$ are all the same, then all datapoints across devices are i.i.d. samples of this distribution, and this process is similar to stochastic gradient descent with batch size $|G_i(t)|$.

*2) Aggregation and synchronization:* The aggregation server $s$ will periodically receive the local estimates $w_i(t)$ from the devices, compute a global update based on these models, and synchronize the devices with the global update. Formally, the $k$th aggregation is computed as

$$w(k) = \frac{\sum_i H_i(k\tau)\cdot w_i(k\tau)}{\sum_i H_i(k\tau)}, \tag{4}$$

where $\tau$ is the fixed aggregation period and $H_i(k\tau) = \sum_{t=(k-1)\tau+1}^{k\tau} |G_i(t)|$ is the number of datapoints node $i$ processed since the last aggregation. Thus, the update is a weighted average factoring in the sample size $H_i(t)$ on which each $w_i(t)$ is based. Once this is computed, each device's local estimate is synchronized, i.e., $w_i(t) \leftarrow w(t/\tau)\forall i$. A lower value of $\tau$ will generally result in faster convergence of $w$, while a higher value requires less network resources. Prior work [5] has already considered how to optimize $\tau$, so we assume it is pre-determined in our formulation, analyzing its effect experimentally in Section V.

### C. Optimization Model for Data Processing Tasks

We now consider the choice of $G_i(t)$, which implicitly defines the computation to be executed by device $i$ at time $t$ for the ML task, i.e., processing all datapoints in $G_i(t)$. There are two possible reasons $G_i(t) \neq D_i(t)$: first, device $i$ may *offload* some of its collected data to another device $j$ or vice versa, e.g., if $i$ does not have enough capacity to process all of the data ($D_i(t) \geq C_i(t)$) or possibly if $j$ has lower computing costs ($c_j(t) \leq c_i(t)$). Second, device $i$ may *discard* data if processing it does not reduce the empirical loss (1) by much. In Figure 2, device 1 offloads or discards all of its data. We collectively refer to discarding and offloading as *data movement*.

*1) Data movement model:* We define $s_{ij}(t) \in [0, 1]$ as the fraction of data collected at device $i$ that is offloaded to device $j \neq i$ at time $t$. Thus, at time $t$, device $i$ offloads $D_i(t)s_{ij}(t)$ amount of data to $j$.[2] Similarly, $s_{ii}(t)$ will denote the fraction

[2]For notational convenience, $D_i(t)$ here refers to the length $|D_i(t)|$, and similarly $G_i(t)$ refers to $|G_i(t)|$. The context will make the distinction clear throughout the paper.

of data collected at time $t$ that device $i$ also processes at time $t$. We suppose that as long as $D_i(t)s_{ij}(t) \leq C_{ij}(t)$, the capacity of the link between $i$ and $j \neq i$, then all offloaded data will reach $j$ within one time interval and can be processed at device $j$ in time interval $t+1$. Since devices must have a link between them to offload data, $s_{ij}(t) = 0$ if $(i,j) \notin E(t)$.

We also define $r_i(t) \in [0,1]$ as the fraction of data collected by device $i$ at time $t$ that will be discarded. In doing so, we assume that device $j$ will not discard data that has been offloaded to it by others, since that has already incurred an offloading cost $D_i(t)s_{ij}(t)c_{ij}(t)$. The amount of data collected by device $i$ at time $t$ and discarded is then $D_i(t)r_i(t)$, and the amount of data processed by each device $i$ at time $t$ is

$$G_i(t) = s_{ii}(t)D_i(t) + \sum_{j \neq i} s_{ji}(t-1)D_j(t-1).$$

In defining the variables $s_{ij}(t)$ and $r_i(t)$, we have implicitly specified the constraint $r_i(t) + \sum_j s_{ij}(t) = 1$: all data collected by device $i$ at time $t$ must either be processed by device $i$ at this time, offloaded to another device $j$, or discarded. We assume that devices will not store data for future processing, which would add another cost component to the model.

*2) Data movement optimization:* We formulate the following cost minimization problem for determining the data movement variables $s_{ij}(t)$ and $r_i(t)$ over the time period $T$:

$$\underset{s_{ij}(t), r_i(t)}{\text{minimize}} \quad \sum_{t=1}^{T} \left( \sum_i G_i(t)c_i(t) + \sum_{(i,j)\in E(t)} D_i(t)s_{ij}(t)c_{ij}(t) + \sum_i f_i(t)L\left(w_i(t)|D_V\right) \right), \tag{5}$$

$$\text{subject to} \quad G_i(t) = s_{ii}(t)D_i(t) + \sum_{j \neq i} s_{ji}(t-1)D_j(t-1), \tag{6}$$

$$s_{ij}(t) = 0, \quad (i,j) \notin E(t), j \neq i, \tag{7}$$

$$r_i(t) + \sum_j s_{ij}(t) = 1, \quad s_{ij}(t), r_i(t) \geq 0, \tag{8}$$

$$G_i(t) \leq C_i(t), \quad s_{ij}(t)D_i(t) \leq C_{ij}(t). \tag{9}$$

Constraints (6–8) were introduced above and ensure that the solution is feasible. The capacity constraints in (9) ensure that the amount of data transferred and processed are within link and node capacities, respectively.

The three terms in the objective (5) correspond to the processing, offloading, and error costs, respectively, as we detail below. We do not include the cost of communicating parameter updates to/from the server in our model; unless a device processes no data, the number of updates stays constant.
*(i) Processing, $G_i(t)c_i(t)$:* This is the computing cost associated with processing $G_i(t)$ of data at node $i$ at time $t$.
*(ii) Offloading, $D_i(t)s_{ij}(t)c_{ij}(t)$:* This is the communication cost incurred from node $i$ offloading data to $j$.
*(iii) Error, $f_i(t)L\left(w_i(t)|D_V\right)$:* This cost quantifies the impact of the data movement on the error of the model at each device $i$.

Note that since $w_i(t)$ is computed as in (3), the model error term is an implicit function of $G_i(t)$, the data processed at device $i$. We include the error from *each* device $i$'s local model at each time $t$, instead of simply considering the error of the final model, since devices may need to make use of their local models as they are updated (e.g., if aggregations are infrequent due to resource constraints [5]).

Discarding data clearly increases the loss, since less data is used to train the ML model. Offloading may also skew the local model if it is updated over a small number of samples $G_i(t)$. We can, however, upper bound the loss function $L(w_i(t))$ regardless of the data movement:

**Theorem 1** (Upper bound on the local loss)**.** *If $L_i(w)$ is convex, $\rho$-Lipschitz, and $\beta$-smooth, if $\eta \leq \frac{1}{\beta}$, and if $L(w(T)) - L(w^\star) \geq \epsilon$ for a suitable lower bound $\epsilon$, then, after $K$ aggregations with a period $\tau$ and defining the constant $\delta_i \geq ||\nabla L_i(w) - \nabla L(w)||$, the local loss will satisfy*

$$L(w_i(t)) - L(w^\star) \leq \epsilon_0 + \rho g_i(t - K\tau), \tag{10}$$

*where $g_i(x) = \frac{\delta_i}{\beta}((\eta\beta+1)^x - 1)$, which implies $g_i(t-K\tau)$ is decreasing in $K$, and $\epsilon_0$ is given by*

$$\frac{1}{t\omega\eta(2-\beta\eta)} + \sqrt{\frac{1}{t^2\omega^2\eta^2(2-\beta\eta)^2} + \frac{Kh(\tau) + g_i(t-K\tau)}{t\omega\eta(1-\beta\eta/2)}}.$$

*Proof:* The full proof is contained in Appendix A of the supplementary material. There, we define $v_k(t)$ for $t \in \{(k-1)\tau, ..., k\tau\}$ as the parameters under centralized gradient descent updates, $\theta_k(t) = L(v_k(t)) - L(w^\star)$, $K = \lfloor t/\tau \rfloor$, and assume $\theta_k(k\tau) \geq \epsilon$ as in [5]. After lower-bounding $\frac{1}{\theta_{K+1}(t)} - \frac{1}{\theta_1(0)}$ and $\frac{1}{L(w_i(t)) - L(w^\star)} - \frac{1}{\theta_{K+1}(t)}$, we can upper-bound $L(w_i(t)) - L(w^\star)$ as

$$\left(t\omega\eta\left(1 - \frac{\beta\eta}{2}\right) - \frac{\rho}{\epsilon^2}\left(Kh(\tau) + g_i(t-K\tau)\right)\right)^{-1} = y(\epsilon).$$

Then, we let $\epsilon_0$ be the positive root of $y(\epsilon) = \epsilon$, which is easy to check exists. The result follows since either $\min_{k \leq K} L(v_k(k\tau)) - L(w^\star) \leq \epsilon_0$ or $L(w_i(t)) - L(w^\star) \leq \epsilon_0$; both imply (10). ∎

In Section IV, we will consider how to use Theorem 1's result to find tractable forms of the loss expression that allow us to solve the optimization (5–9) efficiently and accurately. Moreover, without perfect information on the device costs, capacities, and error statistics over the time period $T$, it is not possible to solve (5–9) exactly, so we will propose methods for estimating them. We will experimentally validate our methodologies in Section V.

## IV. Optimization Model Analysis

We turn now to a theoretical analysis of the data movement optimization problem (5–9). We discuss the choice of error and capacity values under various assumptions (Section IV-A), and then characterize the optimal solution for the ML use cases outlined in Section I (Section IV-B).

### A. Choosing Costs and Capacities

We may not be able to reliably estimate the costs $c_{ij}(t)$, $c_i(t)$, and $f_i(t)$ or capacities $C_i(t)$ and $C_{ij}(t)$ in real time.

Mis-estimations are likely in highly dynamic scenarios of mobile devices, since the costs $c_{ij}(t)$ of offloading data depend on network conditions at the current device locations. Mobile devices are also prone to occasional processing delays called "straggler effects" [19], which can be modeled as variations in their capacities. The error cost, on the other hand, will decrease over time as the model parameters move towards convergence. Here, we propose and analyze network characteristic selection methods. Although these methods also rely on some knowledge of the system, we show in Section V that a simple time-averaging of historical costs and capacities suffices to obtain reasonable data movement solutions.

*1) Choosing capacities:* Over-estimating the device processing capacities will force some data processing to be deferred until future time periods, which may cause a cascade of processing delays. Under- or over-estimations of the link capacities will have similar effects. Here, we formalize guidelines for choosing the capacities in (9)'s constraints so as to limit delays due to over-estimation. As commonly done [37], we assume that processing times on device stragglers follow an exponential distribution $\exp(\mu)$ for parameter $\mu$.

For device capacities, we obtain the following result:

**Theorem 2** (Data processing time with compute stragglers)**.** *Suppose that the service time of processing a datapoint at node $i$ follows $\exp(\mu_i)$, and that $c_{ij}(t)$, $c_i(t)$, $C_i(t)$ are time invariant. We can ensure the average waiting time of a datapoint to be processed is lower than a given threshold $\sigma$ by setting the device capacity $C_i$ such that $\phi(C_i) = \sigma\mu_i/(1+\sigma\mu_i)$, where $\phi(C_i)$ is the smallest solution to the equation $\phi = \exp\left(-\mu_i(1-\phi)/C_i\right)$, which is an increasing function of $C_i$.*

*Proof:* The full proof is contained in Appendix B of the supplementary material. There, we note that the processing at node $i$ follows a D/M/1 queue with an arrival rate $G_i(t) \leq C_i$, and the result follows from the average waiting time in such a queue. ∎

For instance, $\sigma = 1$ guarantees an average processing time of less than one time slot, as assumed in Section III's model. Thus, Theorem 2 shows that we can still (probabilistically) bound the data processing time when stragglers are present.

Network link congestion encountered in transferring data may also delay its processing. Such delays can be handled by carefully choosing the network capacity $C_{ij}(t)$ analogously to Theorem 2's method.

*2) Choosing error expressions:* As shown in Theorem 1, we can bound the local loss at time $t$ in terms of a gradient divergence constant $\delta_i \geq \|\nabla L_i(w) - \nabla L(w)\|$. The following in turn provides an upper bound for $\delta_i$ in terms of $G_i(t)$:

**Lemma 1** (Error convergence)**.** *Suppose that the distributions $\mathcal{D}_i$ have finite second and third moments. Then there exists constants $\gamma_i > 0$ that do not depend on the value of $G_i(t)$ such that*

$$\delta_i \equiv \|\nabla L_i\left(w|G_i(t)\right) - \nabla L(w)\| \leq \frac{\gamma_i}{\sqrt{G_i(t)}} + \frac{\gamma}{\sqrt{|D_V|}} + \Delta, \tag{11}$$

| Use case | Topology | Dynamics |
|---|---|---|
| Smart factories [2] | Hierarchical | Fairly static |
| Connected vehicles [2] | Hierarchical | Rapid changes |
| Augmented reality [3] | Hierarchical, heterogeneous | Rapid changes possible |
| Privacy-sensitive [4], [19] | Social network | Fairly static |

TABLE I: Dominant characteristics of the four fog use cases outlined in Section I-A.

*where $\Delta = ||\nabla L_i(w|D_i) - \nabla L(w|D)||$, $\gamma = \sum_{i=1}^{N}\gamma_i$, and $|D_V|$ is the total number of datapoints generated.*

*Proof:* We can express $||\nabla L_i(w|G_i(t)) - \nabla L(w)||$ as the sum of $||\nabla L_i(w|G_i(t)) - \nabla L_i(w|D_i(t))||$, $||\nabla L(w|D) - \nabla L(w|D_V)||$, and $\Delta$.

We bound $||\nabla L(w|D) - \nabla L(w|D_V)||$ above by $\gamma/\sqrt{|D_V|}$ using the central limit theorem. Since $\nabla L(w|D_V)$ is the average of $\nabla L(w, x_d, y_d)$, $\forall (x_d, y_d) \in D_V$, we can view $\nabla L(w, x_d, y_d)$ as $|D_V|$ samples from a distribution whose expected value is $\nabla L(w|D)$. We repeat this argument for $||\nabla L_i(w|G_i) - \nabla L_i(w|D_i)||$. ∎

The bound in Lemma 1 is rather loose for non-i.i.d data, since, with data movement, the resulting distribution of data at device $i$ will be a mixture of the original distributions of data at each device.

Combining the result in Lemma 1 with Theorem 1, we see that $L(w_i(t)) - L(w^\star) \propto \sqrt{G_i(t)^{-1}}$. Intuitively, as a device processes more data, its loss should decrease. Thus, it is possible to take the error cost $f_i(t)L(w_i(t)|D_V)$ in (5) as $f_i(t)\sqrt{G_i(t)^{-1}}$ with $f_i(t)$ scaling the error importance; $f_i(t)$ may be chosen to decrease over time as the model approaches convergence.

Since $\sqrt{G_i(t)^{-1}}$ is a convex function of $G_i(t)$, with this choice of error cost, (5–9) becomes a convex optimization problem and can be solved relatively easily in theory. When the number of variables is large, however – e.g., if the number of devices $n > 100$ with $T > 100$ time periods, as could be the case in the applications discussed in Section I-A – standard interior point solvers will be prohibitively slow [38]. In such cases, we may wish to approximate the error term with a linear function and leverage faster linear optimization techniques, i.e., to take the error cost as $f_i(t)G_i(t)$ but with $f_i(t) < 0$ since the error decreases when $G_i(t)$ increases. If we neglect the offloaded data $s_{ij}(t)$ for $j \neq i$, we can rewrite this cost as $f_i(t)D_i(t)[1-r_i(t)]$, which is equivalent to minimizing $-f_i(t)D_i(t)r_i(t)$. The error costs from the offloaded data can then be folded into the communication costs $c_{ij}(t)$, and we can approximate the error cost as $-f_i(t)D_i(t)r_i(t)$. Intuitively, discarding data implies a less accurate model, where $f_i(t)$ may be chosen to decrease over time to be consistent with Theorem 1. We experimentally validate such error bounds in Section V.

### B. Optimal Task Distributions

Given a set of costs and capacities for the optimization (5–9), we now characterize the optimal solutions in a range of practical cases. In particular, when we consider a linear error term $f_i(t)r_i(t)D_i(t)$, we have the following result:

**Theorem 3** (Unconstrained resource solution)**.** *Suppose that* $C_i(t) \geq D_i(t) + \sum_{j \in \mathcal{N}_i(t-1)} D_j(t-1)$ *for each device* $i$*, i.e., its compute capacity always exceeds the data it collects as well as any data offloaded to it by its neighbors* $\mathcal{N}_i(t-1) = \{j : (j,i) \in E(t-1)\}$*. Then, if the error cost is linearly approximated as* $f_i(t)D_i(t)r_i(t)$*, the optimal* $s^*_{ij}(t)$ *and* $r^*_i(t)$ *will each be* 0 *or* 1*, with the following conditions for them being* 1 *at node* $i$*:*

$$\begin{cases} s^*_{ik}(t) = 1 & \text{if } c_{ik}(t) + c_k(t+1) \leq \min\{f_i(t), c_i(t)\} \\ s^*_{ii}(t) = 1 & \text{if } c_i(t) \leq \min\{f_i(t), c_{ik}(t) + c_k(t+1)\} \\ r^*_i(t) = 1 & \text{if } f_i(t) \leq \min\{c_i(t), c_{ik}(t) + c_k(t+1)\} \end{cases} \tag{12}$$

*where* $k = \underset{j: j \neq i, (i,j) \in E(t)}{\arg\min} \{c_{ij}(t) + c_j(t+1)\}$.

*Proof:* Since $r_i(t) + \sum_j s_{ij}(t) = 1$ in (8), each datapoint in $D_i(t)$ is either discarded, offloaded, or processed at $i$. It is optimal to choose the option with least marginal cost. ∎

This theorem implies that in the absence of resource constraints, all data a device generates will either be processed, offloaded to the lowest cost neighbor, or discarded. Below, we examine implications of this result for typical fog network topologies.

*1) Fog use cases:* Table I summarizes the topologies of the four fog applications from Section I. Networks in smart factories have fairly static topologies, since they are deployed in controlled indoor settings. They also exhibit a hierarchical structure, with less powerful devices connected to more powerful ones in a tree-like manner, as shown in Figure 1. Connected vehicles have a similar hierarchical structure, with sensors and vehicles connected to more powerful edge servers, but their architectures are more dynamic as vehicles are moving. Similarly, AR applications feature (possibly heterogeneous) mobile AR headsets connected to powerful edge servers. Applications that involve privacy-sensitive data may have very different, non-hierarchical topologies as the links between devices are based on trust, i.e., comfort in sharing private information with a neighboring node. Since social relationships generally change slowly compared to ML model training, these topologies are relatively static.

*2) Hierarchical topologies:* In hierarchical scenarios, more powerful edge servers will likely always have sufficient capacity $C_i(t)$ to handle all offloaded data – thus satisfying the assumptions in Theorem 3 – and they will likely have lower computing costs $c_i(t)$ when compared to other devices. Theorem 3 indicates that, with a linear error cost, sensors would then offload their data to the edge servers, unless the cost of offloading the data exceeds the difference in computing costs. In Section V, we will see from our experiments with our Raspberry Pi testbed that the network cost occasionally exceeds the savings in computing cost from offloading to more powerful devices, in such cases devices process or discard their data.

When the cost of discarding data is nonlinear, the optimal solution is less intuitive: it may be optimal to discard fractions of data if the reduction in error is not worth the additional cost of processing. Formally, in the case of a hierarchical topology, we have the following result:

Fig. 3: Our Raspberry Pi devices running local computations.

**Theorem 4** (Data movement with nonlinear error costs)**.** *Suppose that* $n$ *devices with identical, constant processing costs* $c_j(t) = c$ *and data generation rates* $D_j(t) = D$ *can offload their data to a single edge server, indexed as* $n+1$*. Further assume that there are no resource constraints, that* $c > c_{n+1}$*, that the costs* $c_{ij}(t) = c_t$ *of transmitting to the server are identical and constant, and that the discard cost is given by* $f_i(t)L(w_i(t)) = \gamma/\sqrt{G_i(t)}$ *as in Lemma 1. Then, letting* $s$ *denote the fraction of data offloaded, for* $D$ *sufficiently large, the optimal amount of data discarded as a function of* $s$ *is*

$$r^*(s) = 1 - \frac{1}{D}\left(\frac{\gamma}{2c}\right)^{\frac{2}{3}} - s. \tag{13}$$

*Given the optimal* $r^*$*, the optimal* $s^*$ *is given by*

$$s^* = \frac{1}{nD}\left(\frac{\gamma}{2(c_{n+1} + c_t)}\right)^{\frac{2}{3}}. \tag{14}$$

*Proof:* The full proof is contained in Appendix C in the supplementary materials. There we note that in the hierarchical scenario, the cost objective (5) can be rewritten as

$$n(1-r-s)Dc + nsD(c_{n+1} + c_t) + \frac{n\gamma}{\sqrt{(1-r-s)D}} + \frac{\gamma}{\sqrt{snD}}.$$

Taking the partial derivatives with respect to $r$ and $s$, and noting that a large $D$ forces $r, s \in [0,1]$ gives the result. ∎

Intuitively, as the costs $c$ or $c_{n+1}$ increase, so should the amount of data discarded, as is the case in Theorem 4. Another result of Theorem 4 is that data is neither fully discarded nor fully offloaded, in contrast with Theorem 3. This implies that more exact (nonlinear) error bounds lead to a more balanced distribution of data across nodes in the network.

*3) Social network topologies:* When device networks are larger and have more complex topologies, we can extrapolate from Theorem 3's characterization of individual device behavior to understand data movement in the network as a whole. Consider, for instance, a social network of users in which edges are defined by willingness to share data (Figure 1b). We can find the probability that devices offload data, which allows us to determine the cost savings from offloading:

**Theorem 5** (Value of offloading)**.** *Suppose the fraction of devices with* $k$ *neighbors equals* $N(k)$*. For a social network following a scale-free topology, for example,* $N(k) = \Gamma k^{1-\gamma}$ *for some constant* $\Gamma$ *and* $\gamma \in (2,3)$*. Suppose* $c_i \sim U(0, C)$ *and* $c_{ij} = 0$ *over all time, where* $U(a,b)$ *is the uniform distribution*

*between $a$ and $b$, and that no discarding occurs. Then the average cost savings, compared to no offloading, equals*

$$\sum_{k=1}^{n} N(k)\left(\frac{C}{2}-\frac{C(-1)^k}{k+2}-\sum_{l=0}^{k-1}\binom{k}{l}\frac{C(-1)^l(k+3)}{(l+2)(l+3)}\right). \tag{15}$$

*Proof:* The full proof is contained in Appendix D of the supplementary materials. There, we first find $P_o(k)$ by computing the probability that at least one device $j$ in the neighborhood of $i$ has lower cost than $c_i$, from which we can determine (15). ∎

This result establishes that the reduction in cost from enabling device offloading in such scenarios is approximately linear in $C$: as the range of computing costs increases, there is greater benefit from offloading, since devices are more likely to find a neighbor with lower cost. The processing cost model may for instance represent device battery levels drawn uniformly at random from 0 (full charge) to $C$ (low charge). The expected reduction, however, may be less than the average computing cost $C/2$, as offloading data to another device does not entirely eliminate the computing cost. This is true even in the absence of link costs, e.g., in a social network where nodes have no privacy restrictions on communicating.

We finally consider the case in which resource constraints are present, e.g., for less powerful edge devices. We can find the expected number of devices that will have tight resource constraints:

**Theorem 6** (Probability of resource constraint violation)**.** *Let $N(k)$ be the number of devices with $k$ neighbors, and for each device $i$ with $k$ neighbors, let $p_k(n)$ be the probability that any one of its neighbors $j$ has $n$ neighbors. Also let $\tilde{C}$ denote the distribution of resource capacities, assumed to be i.i.d. across devices, and let $D_i(t) = D$ be constant. Then if devices offload as in Theorem 3, the expected number of devices whose capacity constraints are violated is*

$$\int_{\tilde{C}(x)}\left(\sum_{k=1}^{N} N(k)\mathbb{P}\left[1-P_o(k)+k\sum_{n=1}^{N}\left(\frac{P_o(n)p_k(n)}{n}\right)\geq\frac{x}{D}\right]\right), \tag{16}$$

*with $P_o(k)$ defined as the probability a device with $k$ neighbors offloads its data.*

*Proof:* This follows from Theorem 3, and from obtaining an expression for the expected amount of data that will be processed at a node with $k$ neighbors when offloading is enabled. ∎

Theorem 6 allows us to quantify the complexity of solving the data movement optimization problem when resource constraints are in effect. We observe that it depends on not just the resource constraints, but also on the distribution of computing costs (through $P_o(k)$), since these costs influence the probability devices will want to offload in the first place.

## V. Experimental Evaluation

In this section, we experimentally evaluate our methodology in several scenarios. After discussing the experimental setup in Section V-A, we investigate the performance of network-aware learning in Section V-B. Then, we examine the effects of network characteristics, structure, and dynamics on our methodology in Sections V-C to V-E.

| | Synthetic Costs | | Real Costs | |
|---|---|---|---|---|
| **Framework** | **MLP (%)** | **CNN (%)** | **MLP (%)** | **CNN (%)** |
| Centralized | 92.00 | 98.00 | 92.00 | 98.00 |
| Federated | 89.67 | 96.45 | 89.67 | 96.45 |
| Network-aware | 88.63 | 95.89 | 89.70 | 96.03 |

TABLE II: Network-aware learning achieves within 3% accuracy of both centralized and federated learning on test datasets in all cases.

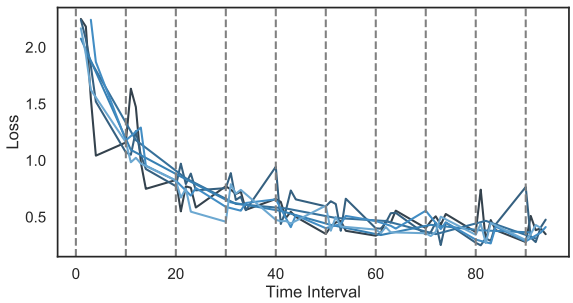



Fig. 4: Training loss over time for each device observed with network-aware learning. The average and variance drop over time as expected.

### A. Experimental Setup

**Machine learning task and models.** We consider image recognition as our machine learning task, using the MNIST dataset [39], which contains 70K images of hand-written digits. We use 60K of these samples as the training dataset $D_V$, and the remainder as our test set. The number of samples $|D_i(t)|$ at node $i$ in period $t$ is modelled using a poisson arrival process with mean $|D_V|/nT$. The $|D_i(t)|$ are then generated by sampling datapoints from $D_V$ uniformly at random and without replacement.

We train multilayer perceptrons (MLP) and convolutional neural networks (CNN) for image recognition on MNIST. Cross entropy loss [40] is used as the loss function $L(w|D_V)$, and a constant learning rate $\eta(t) = 0.01$ is chosen for gradient descent. Unless otherwise stated, results are reported for CNN using $n = 10$ fog devices, an aggregation period $\tau = 10$, and $T = 100$ time intervals.

**Network cost and capacity parameters.** To obtain realistic network costs for nodes $c_i(t)$ and link $c_{ij}(t)$, we collect measurements form a testbed we constructed consisting of six Raspberry Pis as nodes and AWS DynamoDB as a cloud-based parameter server (Figure 3). Three Pis collect data and transmit it over bluetooth to another "gateway" Pi. The three gateway nodes receive this data and can either perform a local gradient update or upload the data to be processed in the cloud. We collect 100 rounds of measurements consisting of gradient update processing times and Pi to DynamoDB communication times while training a two-layer fully connected neural network, with devices communicating over 2.4 GHz WiFi or LTE cellular. These processing times are linearly scaled to

| Setting | Accuracy (%) | Cost | | | | |
|---|---|---|---|---|---|---|
| | | Process | Transfer | Discard | Total | Unit |
| A | 89.72 | 1234 | 0 | 0 | 1234 | 0.265 |
| B | 89.81 | 322 | 120 | 167 | 609 | 0.125 |
| C | 89.54 | 302 | 117 | 160 | 580 | 0.126 |
| D | 83.14 | 336 | 63 | 268 | 667 | 0.136 |
| E | 82.83 | 307 | 46 | 282 | 635 | 0.137 |

TABLE III: Network costs and model accuracies obtained in five different settings. The differences between (A), where no data transfers are permitted, and (B)-(E), which are variants of network-aware learning, show substantial improvements in resource utilization.

range from 0 to 1, and recorded as $c_i(t)$, while the Raspberry Pi to DynamoDB communication times are similarly scaled and saved as $c_{ij}(t)$. For completeness, we also evaluate performance in the case of synthetic costs, where we take $c_{ij}(t), c_i t \sim U(0,1)$ to have comparable ranges to the testbed case. The synthetic/real parameters comparison is included to provide a more thorough comparsion between network-aware learning and state-of-the-arts, since network parameters only affect network-aware learning. Unless otherwise stated, results are reported for the testbed case.

When imposed, the capacity constraints $C_i(t)$ and $C_{ij}(t)$ are taken as the average data generated per device in each time period, i.e., $|D_V|/nT$. The error costs $f_i(t)$ are modeled using the accuracy measurements from historical runs done on the Raspberry Pi testbed. All results are averaged over five iterations.

**Centralized and federated learning.** To see whether our method compromises learning accuracy in considering network costs as additional objectives, we compare against a baseline of centralized ML training where all data is processed at a single device (server). Additionally, we consider the standard implementation of federated learning where there is no data offloading or discarding, i.e., $G_i(t) = D_i(t)$, and aggregations occur after every time interval, i.e. $\tau = 1$ [5].

**Perfect information vs. estimation.** As discussed in Section IV-A, solving (5-9) in practice requires estimating the costs and capacities over the time horizon $T$. To do this, we divide $T$ into $L$ intervals $T_1, ..., T_L$, and in each interval $l$, we use the time-averaged observations of $D_i(t)$, $c_i(t)$, $c_{ij}(t)$, and $C_i(t)$ over $T_{l-1}$ to compute the optimal data movement. The resulting $s^\star_{ij}(t)$ and $r^\star_i(t)$ for $t \in T_l$ are then used by device $i$ to transfer data in $T_l$. This "imperfect information" scheme will be compared with the ideal case in which the network costs and parameters are available (i.e., "perfect informaiton").

### *B. Efficacy of Network-Aware Learning*

Our first experiments investigate the overall efficacy of our method. Here, we report results from a fully connected topology $E(t) = \{(i,j) : i \neq j\}$ among the fog devices; similar results were observed with other configurations.

*1) Model accuracy:* Table II compares the accuracies obtained on the testing datasets by centralized, federated, and network-aware learning on both synthetic and real parameters for both loss functions. The centralized and federated learning algorithms are run until convergence. Our method performs well: it achieves similar accuracy to federated learning and within 3% of centralized learning. The convergence of network-aware learning across individual devices is shown in Figure 4. While some devices experience larger variance in their model loss $L_i(w_i(t))$ over time than others, all devices exhibited an overall rapidly decreasing trend.

Note that network-aware learning produced more accurate models – both in absolute terms and relative to the baselines – when using real rather than synthetic costs. In practical fog environments, see Section I-A, devices with faster gradient computations are also likely to transmit faster. The testbed data and, therefore, real costs introduce such a correlation, and capture the existance of more cost-effective offloading (instead of discarding), which drives model accuracy up.

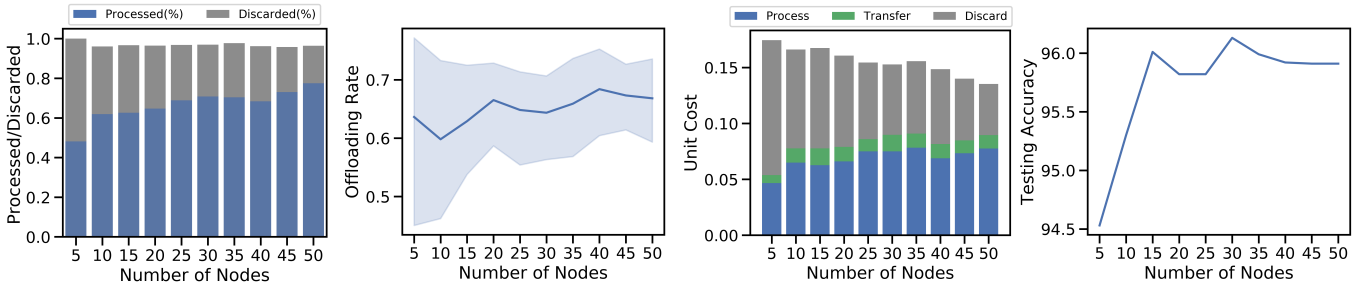
*2) Offloading and imperfect information:* While network-aware learning obtains similar accuracy to federated learning, we expect it will improve network resource costs. Table III compares the costs incurred and model accuracy for five settings, where settings (B) - (E) are applications of network-aware learning.

(A) Offloading and discarding disabled
(B) Perfect information and no capacity constraints
(C) Imperfect information and no capacity constraints
(D) Perfect information and capacity constraints
(E) Imperfect information and capacity constraints.

Each cost component in Table III – process, transfer, and discard – is aggregated over all nodes/links and time periods. The unit cost column is the total cost normalized by the total amount of data generated in that setting, to account for variation in $D_i(t)$ across experiments. Simulations with imperfect information – in settings (C) and (E) – allocate data based on historical network characteristics, which may no longer be accurate, and intuitively could cause unintended large transmission or processing costs. In the capacity limiting cases – settings (D) and (E) – the excess data must be discarded instead, incurring a corresponding discard cost.

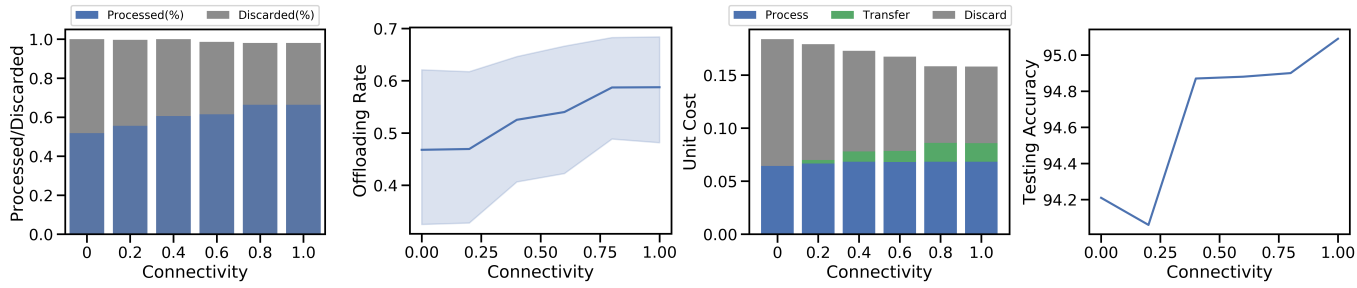
Comparing (A) and (B), we see that allowing data transfers substantially reduces the unit cost– by 53%. The network takes advantage of transfer links, reducing the aggregate processing cost by 74%, by offloading more data. Surprisingly, the accuracy in setting (B) marginally improves on (A) despite some datapoints being discarded: when offloading without capacity constraints, nodes with lower processing costs tend to receive significantly more data, giving them a larger sample size $G_i(t)$ for gradient updates, and thus more accurate parameter estimates that are also more heavily weighted in the aggregations. Even with imperfect information on the parameters in (C), we observe only minor changes in cost or accuracy, highlighting the robustness of the model to estimation errors similar to our observation from the analytics results in Section V-A. The results in (D) and (E) further the point on solution accuracy: when devices have strict capacity constraints, their gradient updates are based on fewer samples, and each node's $L_i(w_i(t))$ will tend to have larger errors.

Overall, while there is a roughly 7% difference in accuracy between (A) and (E), this comes at an improvement of more than 50% in network costs, if higher accuracy was desired for a particular application, the error costs could be more heavily weighted in the objective function (5).




(a) Process vs. discard ratio (b) Data movement rates (c) Unit cost breakdown (d) Testing accuracy

Fig. 5: Impact of the number of nodes $n$ on (a) the ratio of nodes processed vs. discarded, (b) the data movement rate, (c) the unit cost and cost components, and (d) the learning accuracy. The shading in (b) shows the range observed over time periods. We see that network-aware learning scales well with the number of nodes, as the cost incurred per datapoint improves.




(a) Process vs. discard ratio (b) Data movement rates (c) Unit cost breakdown (d) Testing accuracy

Fig. 6: Impact of network connectivity $\rho$ on the different aspects of network-aware learning in Figure 5. Overall, we see that the costs have a linear relationship with $\rho$, that data movement rates increase in response to $\rho$, and that the learning accuracy tends to improve with $\rho$. The shading in (b) indicates the range over time periods.

### C. Effect of Network System Characteristics

Our next experiments investigate the impact of several fog system characteristics on network-aware learning: the number of nodes, $n$, the network connectivity, $\rho$, and the aggregation period, $\tau$. In the experiments for $n$ and $\tau$, the nodes exhibit a fully connected topology, while in the experiment for $\rho$, nodes are connected in a random graph with probability $\rho$, i.e., $P[(i,j) \in E(t), j \neq i] = \rho$.

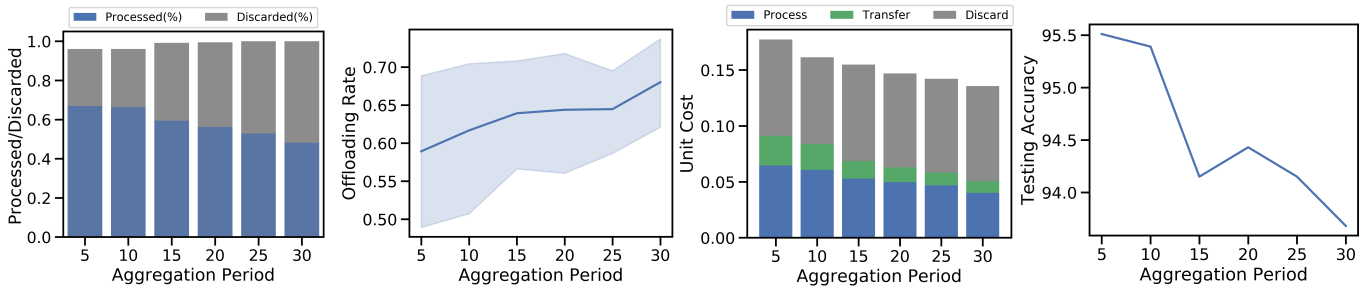
*1) Varying number of nodes $n$:* Figure 5 shows the result of varying the size of the network from $n = 5$ to $n = 50$ in increments of five nodes. Figure 5(a) depicts the change in the fraction of data processed vs. discarded[3]; 5(b) plots the change in the movement rate, i.e., fraction of data that is either offloaded to other devices or discarded; 5(c) provides a detailed breakdown of unit cost by each component; and 5(d) presents the resulting model accuracy.

Overall, we see that our method scales well with the number of nodes, as the unit (i.e. per datapoint) cost in Figure 5(c) steadily decreases with $n$. As the network grows in size, resource-constrained nodes are more likely to have connections to resource-abundant nodes, directly improving the amount of cost-efficient data processing opportunities, which is consistent with Theorems 5 and 6. Figure 5(b) further highlights this trend, as both the minimum offloading rate and the average offloading rate tend to grow with network size. The processing cost now makes up a greater proportion of the total cost, because we are offloading data to be processed at more capable devices instead of discarding, as the discarding rate decreases in Figure 5(a). Even though a larger fraction of data is being processed, the increased processing cost incurred is outweighed by the savings in discard cost in Figure 5(c). As a result of more data being processed, training produces a more accurate ML model in Figure 5(d).

*2) Varying network connectivity $\rho$:* Figure 6 examines the same characteristics in Figure 5 as $\rho$ is varied from 0 (i.e., completely disconnected) to 1 (i.e., fully connected). Overall, we observe a similar trend to the effect of $n$: as connectivity grows, the unit cost per datapoint decreases in Figure 6(c), caused by cheaper alternatives to discarding. The relatively small change in cost as $\rho$ varies between its two extremes indicates that network-aware learning is reasonably robust to connectivity.

More specifically, connectivity produces more opportunities for cost-efficient offloading in Figure 6(b), which increases the total data processed and decreases the total data discarded in Figure 6(a). Discard costs then take a smaller share, relative to processing costs, of the unit costs in Figure 6(c). Intuitively, more data processed leads to a more accurate model in Figure 6(d). Increased network connectivity has a similar effect to having a larger network: there are more connections between resource-constrained nodes and resource-abundant nodes. Network-aware learning then takes advantage of these offloading opportunities to produce cost savings

[3]Rounding used during the solution of the optimization problem resulted in the variance in the sum of the processed and discarded data ratios




(a) Process vs. discard ratio (b) Data movement rates (c) Unit cost breakdown (d) Testing accuracy

Fig. 7: Impact of the aggregation period $\tau$ on the different aspects of network-aware learning in Figure 5. Overall, we see that a higher $\tau$ results in (c) a lower total cost, but also (d) generally a decrease in learning accuracy as well. The shading in (b) indicates the range over time periods.

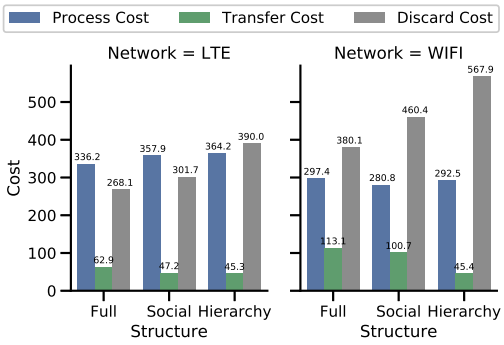



Fig. 8: Cost components for social, hierarchical, and fully connected topologies running network-aware learning on (a) LTE and (b) WiFi network media. Discard costs dominate for each topology in the case of WiFi, while the higher cost factor for LTE depends on the topology.

without compromising model accuracy.

*3) Varying aggregation period $\tau$:* Figure 7 presents the effect of varying the number of local gradient update iterations $\tau$ occuring between global aggregations. Overall, we see that the total unit cost decreases with higher $\tau$, at the cost of decreasing learning accuracy.

Specifically, we expect local models to converge as devices experience longer delays between aggregations, similar to the effects studied in [5]. Devices process their local models on their local datasets for longer, eventually reaching convergence and reducing the value of processing each incremental datapoint. As a result, discarding becomes cost-effective in Figure 7(a), and the discard cost becomes a larger share of the total unit cost in Figure 7(c), with the data movement rate in Figure 7(b) increasing due to this extra discarding. Finally, with a constant number of time intervals $T$, a higher $\tau$ gives a smaller number of total global aggregations $K$, which results in decreased learning accuracy in Figure 7(d), consistent with our findings in Theorem 1.

### D. Effect of Fog Topology

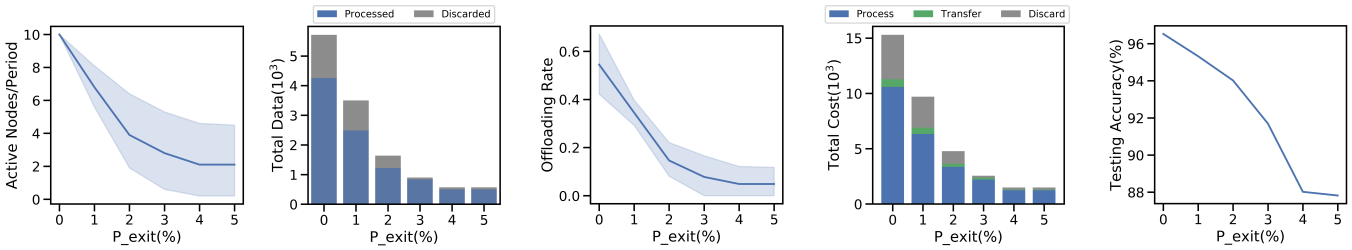
Next, we evaluate network-aware learning on different fog computing topologies. We consider three different graph structures: hierarchical and social network topologies as in Section IV-B, and, for completeness, a fully-connected topology in which all nodes are one hop neighbors. The social network is modelled as a Watts-Strogatz small world graph [41] with each node connected to $n/5$ of its neighbors, and the hierarchical network connects each of the $n/3$ nodes with lowest processing costs to two of the $2n/3$ remaining nodes as leaves, chosen randomly. Our Raspberry Pi testbed provides two different wireless network media for which we compare the costs: LTE and WiFi.

The results are shown in Figure 8. Both media exhibit similar trends across the topologies. Note that network topology determines the availability of cost-effective offloading opportunities: the fully-connected topology maximizes the degree of each node, while the hierarchical topology minimizes the average degree. As the average degree becomes smaller, desirable offloading opportunities tend to become scarcer, leading to decreases in transfer cost and increases in processing and/or discard costs, as transferred data becomes locally processed or discarded instead. This can explain the observation of the transfer cost decreasing and the discard cost increasing as the topology is changed from full to hierarchy in both cases.

One major difference between the media is that WiFi skews more heavily towards discarding. WiFi has fewer interference mitigration techniques than cellular, so, in the presence of several devices, we expect its links to exhibit longer delays. Consequently, both the discard and transfer costs are larger for WiFi than their LTE counterparts for the fully-connected, social, and hierarchical network structures. Results in both cases are consistent with the findings from varying the network connectivity in Figure 6 too: as networks become less connected and edges grow sparse, the ability of individual devices to offload their data to lower cost alternatives diminishes. Devices will marginally increase their data processing workloads, but ultimately a significant fraction of the data is discarded.

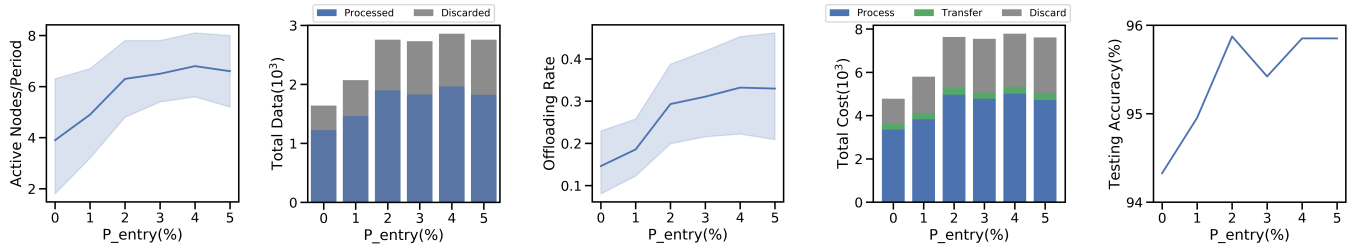
### E. Effect of Dynamic Networks

Finally, we consider network-aware learning on dynamic networks, i.e., when nodes are entering and exiting the network during model learning. Initially, at time $t = 0$, each device is in the network. Then, at each $t$, each device currently in the network will exit with probability $p_{exit}$, and each device not in the network will re-enter with probability $p_{entry}$. For a worst-case analysis, we assume that nodes cannot transmit their local




(a) Average active nodes (b) Process vs. discard (c) Data movement rates (d) Total cost breakdown (e) Testing accuracy

Fig. 9: Impact of increases in the probability of node exits on data movement and costs with probability of node re-entry at the worst-case of 0%. The shading in (a) and (c) indicates the range over time periods. When the average active nodes per period drops by 80%, network-aware learning adapts to transferring and discarding almost no samples, which results in an impact of only 8% on the trained model.




(a) Average active nodes (b) Process vs. discard (c) Data movement rates (d) Total cost breakdown (e) Testing accuracy

Fig. 10: Impact of increases in the probability of node re-entry in each time period, with the probability of node exits fixed at 2%. The shading in (a) and (c) indicates the range over time periods. Compared with Figure 9, we see that node exits impact the different aspects of network-aware learning more than node re-entries.

| Setting | Acc(%) | Nodes | Cost | | | |
|---|---|---|---|---|---|---|
| | | | Process | Transfer | Discard | Unit |
| Static | 95.83 | 10 | 399 | 66 | 328 | 0.135 |
| Dynamic | 94.79 | 7.8 | 300 | 56 | 256 | 0.144 |

TABLE IV: Comparison of network-aware learning characteristics on static and dynamic networks, with the probability of nodes entering and exiting fixed at 1%. "Acc" represents model accuracy and "Nodes" represents the average number of active nodes per aggregation period. Overall, we see that node churn of 20% has an impact of roughly 6% on unit costs, and 1% on accuracy.

update results just prior to exiting the network, and that nodes must wait for the ongoing aggregation period to finish when they initially re-join the network in order to obtain the current global parameters.

Table IV shows the effect of network dynamics with $p_{exit} = p_{entry} = 1\%$ compared with the static case. In a dynamic network, network-aware learning operates with less data and compute capacity available for optimization as the number of active nodes per aggregation period decreases from 10 in the static case to an average of 7.8 in the dynamic case. Node exits always result in at least one inactive node - even if a new node enters, it must wait for the synchronized global parameters. Still, even a 20% churn in active nodes per time period only leads to a 6% jump in unit costs incurred per datapoint due to the fewer opportunities available for data movement, indicating that network-aware learning is reasonably robust to network dynamics. Although the total numerical costs decrease across all columns in Table IV (since there is less data available overall), the fraction of costs due to discarding grows while the fraction due to processing falls. Less processed data ultimately results in a model with slightly poorer accuracy, which drops by roughly 1%.

Next, we study the impact of varying the probability of node exit and node re-entry, respectively.

*1) Varying probability of node exit $p_{exit}$:* Figure 9 shows the results with $p_{exit}$ varied from 0% to 5% in increments of 1% and $p_{entry}$ fixed at 0%. Figure 9a gives the variation in average active nodes per period, and the remaining four sub-figures display the different aspects of network-aware learning as in Figures 5-7.

Figure 9a depicts a sharp decline in the number of active nodes per period as $p_{exit}$ grows - the solid line is the average active nodes per period while the shaded area shows the range of active nodes per period. At 5% $p_{exit}$, the network has at most five active nodes/period, which is only half of the total nodes.

Both total generated data and total cost decrease sharply in Figures 9b and 9d, respectively: since the network has fewer active nodes, there is less data overall and consequently the total cost will be smaller. The ratio of processed versus discarded data in Figure 9b indicates that more data is processed as the likelihood of node exit increases, which in turn reduces the fraction of cost incurred due to discarding in Figure 9d: a high exit probability slows convergence speed and, as per the analysis following Lemma 1, results in expensive discard costs. Due to both high discard costs and fewer opportunities for offloading, average data movement rate drops from 0.6 to 0.1 in Figure 9c. The resulting machine learning model declines in testing accuracy by roughly 8% in Figure 9e as a result of less data being available to train the models. Still, this

decrease of 8% comes with 80% less nodes being active per time period on average, showing that network-aware learning is able to adapt to such dynamics.

*2) Varying probability of node entry $p_{entry}$:* Figure 10 shows the results when $p_{entry}$ is varied from 0% to 5% and $p_{exit}$ is fixed at 2%.

The average active nodes per period increases with probability of node entry until 4% in Figure 10a, where $p_{entry}$ is twice as large as $p_{exit}$. Active nodes generate data and serve as possible offloading destinations. Consequently, there is growth in both total data generated in Figure 10b and average data movement rate in Figure 10c. The growth in total data directly results in higher total costs in Figure 10d, which is driven mainly by increases in processing and discard costs. The availability of more low-cost offloading opportunities and more nodes with processing capabilities allows more data to be processed in network-aware learning, ultimately helping the network produce a high quality machine learning model in Figure 10e that approaches the 96% accuracy achieved in the absence of network dynamics.

Comparing Figures 9 and 10, it seems that node exit impacts network-aware learning more heavily than node re-entry. For instance, in Figures 9b and 10b, node exit appears to influence the total data processed and discarded more substantially. Since our experiments assume "worst-case" effects of network dynamics, node re-entries experience an idle period and are effectively inactive until the next aggregation begins. Therefore, for preserving model quality in network-aware learning, preventing node exits is more important than promoting entry, at least in small edge networks, such as our example with at most ten nodes. In large scale networks, a large $p_{entry}$ may have a more significant impact.

## VI. Conclusion and Future Work

To the best of our knowledge, this was the first work to distribute ML training tasks over devices in a fog computing network while considering the compute-communication tradeoffs inherent in fog scenarios. We developed a framework to optimize the distribution of training tasks on a fog network, taking into account both physical computing and communication costs and the error achieved by the models at each device. We derived new error bounds when devices can transfer their local data processing to each other, and theoretically bounded the impact of these transfers on the cost and accuracy of the model training. Through experimentation with a popular machine learning task, we showed that our network-aware scheme significantly reduces the cost of model training while achieving comparable accuracy to the recently popularized federated learning algorithm for distributed training, and is robust to changes in network characteristics and dynamics.

Our framework and analysis point to several possible extensions. First, while we do not observe significant heterogeneity in compute times on our wireless testbed, in general fog devices may experience compute straggling and failures, which might benefit from more sophisticated offloading mechanisms. Second, predicting devices' mobility patterns and the resulting network connectivity can likely further optimize the data offloading. Finally, for some applications, one might wish to learn individual models for each device, which would introduce new performance tradeoffs between offloading and data processing.

## Acknowledgment

This work was partially supported by NSF CNS-1909306, by the Army Research Office under grant W911NF1910036, and by Northrup Grumman under Year 11 grants.

## References

[1] Y. Tu, Y. Ruan, S. Wagle, C. Brinton, and C. Joe-Wong, "Network-Aware Optimization of Distributed Learning for Fog Computing," in *IEEE Conference on Computer Communications (INFOCOM)*, 2020.

[2] Cisco Systems, "Demystifying 5G in Industrial IOT," White Paper, 2019. [Online]. Available: https://www.cisco.com/c/dam/en_us/solutions/iot/demystifying-5g-industrial-iot.pdf

[3] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile Augmented Reality Survey: From where we are to where we go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.

[4] K. Rao, "The Path to 5G for Health Care," IEEE Perspectives on 5G Applications and Services. [Online]. Available: https://futurenetworks.ieee.org/images/files/pdf/applications/5G--Health-Care030518.pdf

[5] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive Federated Learning in Resource Constrained Edge Computing Systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[6] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[7] IEEE Spectrum, "Applications of Device-to-Device Communication in 5G Networks," White Paper. [Online]. Available: https://spectrum.ieee.org/computing/networks/applications-of-devicetodevice-communication-in-5g-networks

[8] M. Somisetty, "Big Data Analytics in 5G," IEEE Perspectives on 5G Applications and Services. [Online]. Available: https://futurenetworks.ieee.org/images/files/pdf/applications/Data-Analytics-in-5G-Applications030518.pdf

[9] S. Pu, W. Shi, J. Xu, and A. Nedić, "A Push-Pull Gradient Method for Distributed Optimization in Networks," in *IEEE Conference on Decision and Control (CDC)*, 2018, pp. 3385–3390.

[10] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[11] T.-Y. Yang, C. Brinton, P. Mittal, M. Chiang, and A. Lan, "Learning Informative and Private Representations via Generative Adversarial Networks," in *IEEE International Conference on Big Data*. IEEE, 2018, pp. 1534–1543.

[12] S. A. Ashraf, I. Aktas, E. Eriksson, K. W. Helmersson, and J. Ansari, "Ultra-Reliable and Low-Latency Communication for Wireless Factory Automation: From LTE to 5G," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–8.

[13] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and Stale Gradients Can Win the Race: Error-Runtime Trade-offs in Distributed SGD," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018, pp. 803–812.

[14] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[15] R. Shokri and V. Shmatikov, "Privacy-Preserving Deep Learning," in *ACM Conference on Computer and Communications Security (SIGSAC)*, 2015, pp. 1310–1321.

[16] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascn, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konen, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. zgr, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T.

Suresh, F. Tramr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and open problems in federated learning," 2019.

[17] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated Learning with Non-IID Data," *arXiv:1806.00582*, 2018.

[18] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated Multi-Task Learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 4424–4434.

[19] G. Neglia, G. Calbi, D. Towsley, and G. Vardoyan, "The Role of Network Topology for Distributed Machine Learning," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019, pp. 2350–2358.

[20] J. Konen, H. B. McMahan, F. X. Yu, P. Richtrik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016.

[21] N. Strom, "Scalable distributed dnn training using commodity gpu cloud computing," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[22] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017. [Online]. Available: http://dx.doi.org/10.18653/v1/D17-1045

[23] F. Sattler, S. Wiedemann, K.-R. Mller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," 2019.

[24] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with asynchronous model update and temporally weighted aggregation," 2019.

[25] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar, "Peer-to-peer federated learning on graphs," 2019.

[26] N. H. Tran, W. Bao, A. Zomaya, M. N. H. Nguyen, and C. S. Hong, "Federated Learning over Wireless Networks: Optimization Model Design and Analysis," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019, pp. 1387–1395.

[27] T. Chang, L. Zheng, M. Gorlatova, C. Gitau, C.-Y. Huang, and M. Chiang, "Demo: Decomposing Data Analytics in Fog Networks," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2017.

[28] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics," in *IEEE Conference on Computer Communications (INFOCOM)*, 2018, pp. 1421–1429.

[29] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019, pp. 1423–1431.

[30] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed Deep Neural Networks over the Cloud, the Edge and End Devices," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 328–339.

[31] X. Xu, D. Li, Z. Dai, S. Li, and X. Chen, "A heuristic offloading method for deep learning edge services in 5g networks," *IEEE Access*, vol. 7, pp. 67 734–67 744, 2019.

[32] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, pp. 96–101, 2018.

[33] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2020.

[34] T. Yang, "Trading Computation for Communication: Distributed Stochastic Dual Coordinate Ascent," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2013, pp. 629–637.

[35] Y. Zhang, J. Duchi, M. I. Jordan, and M. J. Wainwright, "Information-theoretic Lower Bounds for Distributed Statistical Estimation with Communication Constraints," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2013, pp. 2328–2336.

[36] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[37] F. Farhat, D. Z. Tootaghaj, Y. He, A. Sivasubramaniam, M. Kandemir, and C. R. Das, "Stochastic Modeling and Optimization of Stragglers," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1164–1177, 2016.

[38] F. M. F. Wong, Z. Liu, and M. Chiang, "On the Efficiency of Social Recommender Networks," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 2512–2524, 2016.

[39] Y. LeCun, C. Cortes, and C. J. C. Burges, "The MNIST Database of Handwritten Digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[40] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[41] M. Chiang, *Networked Life: 20 Questions and Answers*. Cambridge University Press, 2012.