Network-Aware Optimization of Distributed Learning for Fog Computing

Yuwei Tu[‡], Yichen Ruan[†], Satyavrat Wagle[†], Christopher G. Brinton[*], and Carlee Joe-Wong[†]
[*]Purdue University, [†]Carnegie Mellon University, [‡]Zoomi Inc.
[*]cgb@purdue.edu, [†]{yichenr, srwagle, cjoewong}@andrew.cmu.edu

***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 are (i) heterogeneity in devices' compute resources and (ii) topology constraints on which devices can communicate. We are the first to address these challenges by developing a network-aware distributed learning optimization methodology where devices process data for a task locally 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, 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 a device offloading is approximately linear in the range of computing costs in the network. Our subsequent experiments on both synthetic and real-world datasets we collect confirm that our algorithms are able to improve network resource utilization substantially without sacrificing the accuracy of the learned model.**

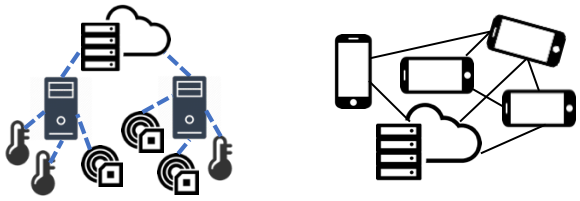
***Index Terms*—federated learning, offloading, fog computing**

I. Introduction

New technologies like autonomous cars and smart factories are coming to rely extensively on data-driven machine learning (ML) algorithms [1]–[3] 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 [4].

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 [5]. 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. Though centralized ML algorithms are not optimized for such environments, data analytics is expected to be a major driver of 5G adoption [6].

Initial efforts in decentralizing ML have focused on decomposing model parameter updates over several nodes, typically managed by a centralized serving entity [7], [8]. 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 or mobility. Figure 1 illustrates two example fog topologies. A central question that arises, then, in adapting ML methodologies to these 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) 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.

A. Machine Learning in Fog Environments

ML models are generally trained by iterating over a dataset to estimate parameter values (e.g., weights in a neural network) that best "fit" the empirical data. We face two major challenges in adapting such training to fog environments: *(i) heterogeneity in devices' compute resources* and *(ii) constraints on devices' abilities to communicate with each other*. We outline these characteristics in some key applications below:

Privacy-sensitive applications. Many ML applications learn models on sensitive user data, e.g., for health monitoring [3]. Due to privacy concerns, most of these applications have devices train their models on local data to avoid revealing data to untrustworthy nodes [9]. Offloading 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 [1], 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 [2] 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 within factory production lines to communicate across the factory floor [1], [10], in turn enabling distributed ML algorithms to use this data for, e.g., predicting production delays. Our work can 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

We first differentiate our work from related literature in Section II. 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 parameters 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 parameters 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 computing overhead yet achieves an accuracy comparable to centralized model training.

Due to page limitations, proofs are abbreviated as sketches; full versions are available in the technical report [11].

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. 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 [8], [12], [13]. Such a framework preserves user privacy by keeping data at local devices [14] and reduces the communication between devices and the central server.

Federated learning introduces two major new challenges: first, since each device generates the data it analyzes, the data may not be identically or independently distributed across the different devices. Second, since the local devices are typically resource-constrained, they may have limited ability to rapidly compute gradient values and communicate with the aggregation server. Many works have attempted to address the first challenge, e.g., through sharing user data [15] or training user-specific models [16]. When the devices attempt to learn a single model, recent efforts have considered optimizing the frequency of parameter aggregations according to available network and computing resources [4], or adopting a peer-to-peer framework in which parameter updates are shared with neighboring devices instead of a central server [17]. However, these works do not optimally distribute parameter computations between devices, and do not consider the compute-communication tradeoffs inherent in fog scenarios.

Offloading computations from constrained mobile devices to nearby edge servers when there is a high-bandwidth connection between them will intuitively improve system performance, and has been shown to significantly accelerate training of a linear regression model [18] and inference on a neural network model [19]. Other works have considered splitting deep neural network layers between fog devices and an edge server for faster inference [20], [21]. We instead consider generic ML frameworks, and additionally provide theoretical performance bounds not found in these prior works.

III. Model and Optimization Formulation

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

A. Fog Computing System Model

Fog computing nodes. We consider a set V of n devices, an aggregation server s, and discrete time intervals $t = 1, \ldots, T$. Each device, e.g., a sensor or smartphone, can both collect data and process it to contribute to an 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.

Data collection and processing. We use $D_i(t)$ to denote the set of data collected by device $i \in V(t)$ at time t;

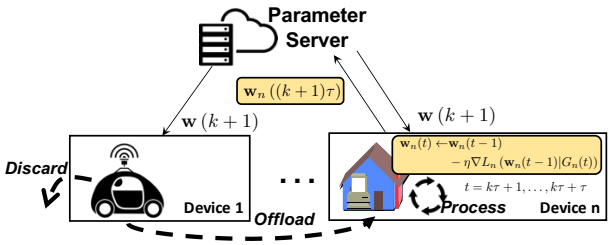


Fig. 2: Federated learning updates between aggregations k and $k+1$. Device 1 discards all of its data or offloads it to device n, which computes τ gradient updates on its local data. The final parameter values are averaged at the parameter server, with the result sent back to the devices to begin a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

$d \in D_i(t)$ denotes each datapoint. (We may have $D_i(t) = 0$ if a device does not collect data.) $G_i(t)$, by contrast, denotes the set of datapoints *processed* by each device at time t; our optimization in Section III-C relates $G_i(t)$ to the datasets $D_i(t)$. In conventional learning frameworks, $D_i(t) = G_i(t)$, as all devices process the data they collect [4]; 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, incurring a cost of $c_i(t)$ for each point. This cost and capacity may for instance represent the battery level; devices with low battery will have lower capacities $C_i(t)$ and higher costs $c_i(t)$.

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 the server. Note that the scenarios outlined in Section I-A each possess such an architecture: in smart factories, for example, a subset of the floor sensors connect to each controller. Each link $(i,j) \in E(t)$ is characterized by a capacity $C_{ij}(t)$, i.e., the maximum datapoints it can transfer, 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 higher if sending from i to j is less desirable at t.

Data structure. Each datapoint d can be represented as (x_d, y_d), where x_d is an attribute/feature vector and y_d is an associated label for model learning. 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 [22], [23] 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 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 ML applications, e.g., image recognition from cameras at fixed locations or AR users with random mobility patterns. We will use such an image dataset for evaluation in Section V.

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 depends on the ML model (e.g., squared error for linear regression, cross-entropy loss for multi-class classifiers [24]). 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 [8].

Fog computing allows (1) to be solved distributedly: instead of computing the solution at the server s, we can use computations at each device i. Below, we follow the commonly used federated averaging framework [4] in specifying these local computations and global aggregation, 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 [8]. 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 average the local estimates $w_i(t)$ from the devices and synchronize the devices with a global update. Formally, the kth 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 τ 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 on which each $w_i(t)$ is based. Once this is computed, each device's local estimate is synchronized, i.e., $w_i(t) \leftarrow w(t/\tau)$. A lower value of τ will result in faster convergence of w, while a higher value requires less network resources. Prior work [4] has considered how to optimize τ, 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 ML tasks to be executed by device i at time t, 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 ($D_i(t) \geq C_i(t)$) 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*. 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.

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.[1] Similarly, $s_{ii}(t)$ will denote the fraction 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 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:

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

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

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

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

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

Constraints (6–8) were introduced above and ensure that the solution is feasible. The capacity constraints in (9) ensure that the amounts 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.

(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), it 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 [4]). 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, ρ-Lipschitz, and β-smooth, if $\eta \leq \frac{1}{\beta}$, and if $L(w(T)) - L(w^\star) \geq \epsilon$ for some $\epsilon > 0$, then after K aggregations with a period τ and defining the constant $\delta_i \geq ||\nabla L_i(w) - \nabla L(w)||$,*

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

where $g_i(x) = \frac{\delta_i}{\beta}((\eta\beta+1)^x - 1)$ implying $g_i(t-K\tau)$ is decreasing in K, and ϵ_0 is given by

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

[1]For notational convenience, $D_i(t)$ here refers to the length $|D_i(t)|$, and similarly for $G_i(t)$ in this section. The context makes the distinction clear.

Proof: 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 $\theta_k(k\tau) \geq \epsilon$ [4]. 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

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

Let ϵ_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 (5–9) efficiently and accurately. Indeed, 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. We will experimentally validate our proposed methods for estimating these parameters 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 parameters 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 Cost Parameters and Capacities

We may not be able to reliably estimate the cost parameters $c_{ij}(t)$, $c_i(t)$, 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" [17], which can be modeled as variations in their capacities. The error cost, on the other hand, will change over time as the model parameters move towards convergence. Here, we propose and analyze parameter selection methods.

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 [25], we assume that processing times on device stragglers follow an exponential distribution $\exp(\mu)$ for parameter μ.

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 σ by setting the device capacity parameter 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 $\phi = \exp(-\mu_i(1-\phi)/C_i)$, which is an increasing function of C_i.*

Proof: 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 in transferring data may also delay its processing. Such delays can be handled by carefully choosing the network capacity analogously to Theorem 2's method; details can be found in our technical report [11].

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 δ_i in terms of $G_i(t)$:

Lemma 1 (I.i.d. error convergence)**.** *Suppose that the distributions $\mathcal{D}_i$ are identical so that $\mathcal{D}_i = \mathcal{D}$, and that $\mathcal{D}$ has finite second and third moments. Then there exists a constant $\gamma > 0$ that does not depend on the value of $G_i(t)$ such that*

$$\delta_i \equiv \|\nabla L_i\left(w|G_i(t)\right) - \nabla L(w)\| \leq \frac{\gamma}{\sqrt{G_i(t)}} \qquad (11)$$

Proof: The result follows immediately from the central limit theorem upon viewing each $\nabla l(w, x_d, y_d)$ as a sample from the distribution induced by $\nabla l(w|\mathcal{D})$. ∎

Assuming i.i.d. data distributions is reasonable for many ML applications: for instance, AR users might follow statistically similar mobility patterns throughout an area, and sensors on a factory floor might monitor machines with similar failure patterns. Combining the result in Lemma 1 with Theorem 1, we find that $L(w_i(t)) - L(w^\star) \propto \sqrt{G_i(t)^{-1}}$. 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 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 – standard interior point solvers will be prohibitively slow [26]. 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 $-f_i(t)r_i(t)$. 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.

B. Optimal Task Distributions

Given a set of cost and capacity parameters for the optimization (5–9), we now characterize the optimal solutions in

Use case	Topology	Dynamics
Smart factories [1]	Hierarchical	Fairly static
Connected vehicles [1]	Hierarchical	Rapid changes
Augmented reality [2]	Hierarchical, heterogeneous	Rapid changes possible
Privacy-sensitive [3], [17]	Social network	Fairly static

TABLE I: Dominant characteristics of the four use cases.

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 conditions for 1 at node i being:*

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

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

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 topologies.

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. Since social relationships generally change slowly compared to ML model training, these topologies are relatively static.

1) Hierarchical topologies: In hierarchical scenarios, more powerful edge servers will likely always have sufficient capacity $C_i(t)$ to handle all offloaded data (satisfying the assumptions in Theorem 3), and they will likely experience lower computing costs $c_i(t)$ as well 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 costs of offloading the data exceed the difference in computing costs. In Section V, we will see on our Raspberry Pi testbed that the network cost does indeed sometimes exceed the gain in computing cost from offloading to more powerful devices.

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:

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, $c > c_{n+1}$, the costs $c_{ij}(t) = c_t$ of transmitting to the server are identical and constant, and 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: 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, increases in the costs c_{n+1}, c_t, data D, or devices n should cause the amount of data offloaded to decrease and the amount discarded to increase. D has the strongest effect: with more data at each node, the fraction needed for processing at the server and devices to manage the discard cost decreases inversely. The only cost that impacts r but not s is c, as it is a device parameter that does not involve the network.

2) 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 a given device offloads 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 $N(k)$, e.g., in a scale-free network $N(k) = \Gamma k^{1-\gamma}$ for some constant Γ and $\gamma \in (2,3)$. Suppose $c_i \sim U(0,C)$ and $c_{ij} = 0$, where $U(a,b)$ is the uniform distribution between a and b, with no discarding. Then the average cost savings, compared to no offloading, equals*

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

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

The processing cost model may for instance represent device battery levels drawn uniformly at random from 0 (full charge) to C (low charge). 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 in a scale-free topology, there is greater benefit from offloading, since devices are more likely to find a neighbor with lower cost. The expected reduction, however, may be less than the average computing cost $C/2$ even in the absence of link costs, as offloading data to another device does not entirely eliminate the computing cost.

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 with 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 its neighbor 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 determining 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. We discuss the general setup in Section V-A, and present our results in Sections V-B to V-D.

A. Experimental Setup

Data samples and ML models. We consider the MNIST dataset [27], which contains 70K images of hand-written digits. We use 60K of them as the training dataset D_V, and the remainder as our test set. Each node i is randomly allocated $|D_i(t)| \sim U(0, |D_V|/(nT))$ datapoints from D_V, without replacement. We train a multilayer perceptron (MLP) and a convolutional neural network (CNN) for image recognition on MNIST, with cross entropy loss [28] as the loss function $L(w|D_V)$, and constant learning rate $\eta(t) = 0.01$ in (3). Unless otherwise stated, results are reported for CNN, an aggregation period $\tau = 10$ in (4), and $T = 100$ time intervals.

Method	MLP	CNN
Centralized learning	92.58%	98.39%
Federated learning	90.33%	96.81%
Network-aware learning	90.34%	96.49%

TABLE II: Our method, centralized learning, and federated learning show comparable accuracies on the test dataset.

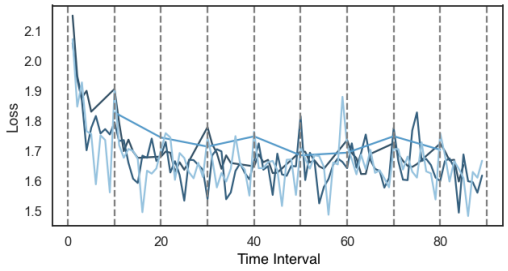


Fig. 4: Training loss over time for each device observed with our method. The average and variance drop over time as expected.

Cost and capacity parameters. In the default case, we simulate $n = 10$ fog devices and one server. When imposed, the capacity constraints $C_i(t)$ and $C_{ij}(t)$ are drawn from $U(0, 4 \max D_i(t))$ and $U(0, 4 \max D_i(t)/n)$ respectively. Due to randomization, results are averaged over several iterations.

We consider both real and synthetic cost parameters. For the synthetic parameters, $c_{ij}(t) \sim U(0, 1/n)$ and $c_i(t) \sim U(0, 1)$. The real-world measurements come from our testbed consisting of six Raspberry Pis using AWS DynamoDB as a cloud-based parameter server (Figure 3). Three Pis each 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 (processing time recorded as $c_i(t)$) or upload the data to DynamoDB (communication time recorded as $c_{ij}(t)$) to be processed there. We collect 100 sets of processing and communication measurements while training a two-layer fully connected neural network, with devices communicating over 2.4 GHz WiFi or LTE cellular.

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., $\tau = 1$) [4].

Perfect information vs. estimation. As discussed in Section IV-A, solving (5-9) in practice requires estimating the cost and capacity parameters over the time horizon T. To do this, we divide T into L intervals $T_1, ..., T_L$, and in each interval l, we

Setting	Accuracy	Cost				
		Process	Transfer	Discard	Total	Unit
A	82.89%	2078	0	0	2078	0.48
B	90.24%	367	620	166	1153	0.26
C	89.24%	326	606	67	999	0.23
D	82.10%	225	835	14	1074	0.24

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

use the time-averaged observations of $D_i(t)$, $p_i(t)$, $c_{ij}(t)$, 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 be used by device i to transfer data in T_l. This "imperfect information" scheme will be compared with the ideal case of perfect information.

B. Efficacy of Network-Aware Learning

Our first experiments seek to establish the overall efficacy of our method. Here, we use the synthetic cost parameters with a fully connected topology $E(t) = \{(i,j) : i \neq j\}$; qualitatively similar results were observed with other configurations.

Model accuracy. Table II compares the average accuracy on the testing datasets obtained by the MLP and CNN models trained with centralized, federated, and network-aware learning, where the centralized and federated cases are run until convergence. Our method does well: it matches federated learning's accuracy, and does only 2% worse than centralized learning. We also plot the training loss $L_i(w_i(t))$ across devices over time in Figure 4, confirming that while some devices have higher local losses, all tend to decrease over time.

Cost reduction with imperfect information. Table III compares the costs incurred and model accuracy for four settings: (A) offloading and discarding disabled, (B) network-aware learning with perfect information and no capacity constraints, (C) network-aware learning with imperfect information and no capacity constraints, and (D) network-aware learning with imperfect information and capacity constraints. Each cost is aggregated over nodes/links and time periods. The unit cost normalizes the total cost over the amount of data generated in that setting, to account for the $D_i(t)$ varying randomly.

Comparing (A) and (B), we see that allowing data transfers substantially reduces the unit cost– by 46%. Surprisingly, the accuracy also improves from (A) to (B) 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 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 observations from the analytics results in Section IV-A. The result for (D) furthers 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. However, the total cost of (D) is still significantly lower than (A) with a comparable accuracy.

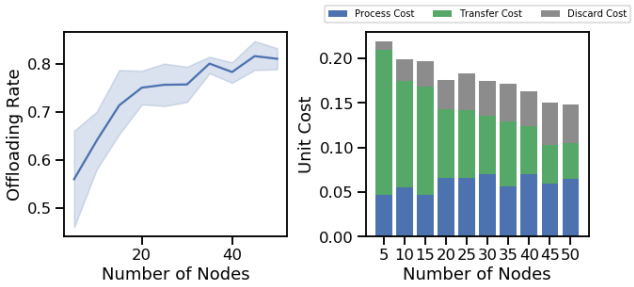


Fig. 5: Impact of the number of nodes n on (a) the average offloading rate and (b) the different cost components. The shading in (a) shows the range over time periods. We see that our method scales well with the number of nodes, as the cost incurred per datapoint improves.

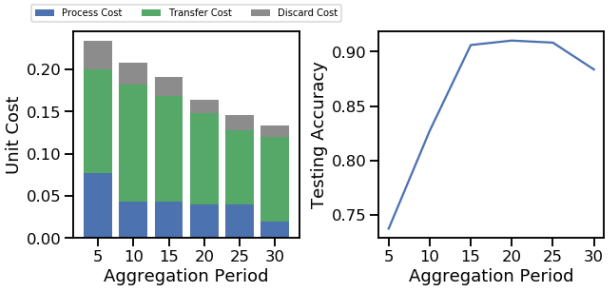


Fig. 6: Impact of the global aggregation period τ on (a) the cost components and (b) the learning accuracy. Increasing τ causes the total costs to decrease, but only improves learning accuracy to a point, illustrating a tradeoff between synchronicity and error convergence.

C. Effect of Network System Parameters

Our next experiments on synthetic cost data assess the impact of n, the number of nodes, and the aggregation period, τ, on a fully connected topology. Then, we consider the effect of connectivity when nodes are connected in a random graph with probability ρ, i.e., $P[(i,j) \in E(t), j \neq i] = \rho$.

Varying number of nodes n. Figure 5 shows the (a) offloading rate and (b) unit costs as the network size changes. Overall, we see that our method scales well with the number of nodes, as the unit (i.e., per datapoint) cost steadily decreases with n. The transferring (i.e., offloading) cost drives this improvement, as the processing and discarding costs actually increase: with more nodes, the network leverages lower cost opportunities for offloading – hence the increase in offloading rate to over 80% when $n = 50$ – as long as it is less than any increase in processing cost, consistent with Theorem 3. The learning accuracy increases slightly from 88 to 92% as n increases.

Varying aggregation period τ. Figure 6 shows the (a) unit costs and (b) learning accuracy as the period of global aggregation is varied. A larger value of τ yields smaller total unit cost, but only improves learning accuracy until roughly $\tau = 20$: below this, the nodes are not processing enough datapoints in-between aggregations for the local parameters to become steady, and above it, the local models are not synchronized frequently enough. We also note that the cost breakdown exhibits a different trend than in Figure 5, as the transmission cost stays relatively constant while the processing and discarding costs decrease: with a longer τ, data can be discarded with a lower cost towards the end of each period.

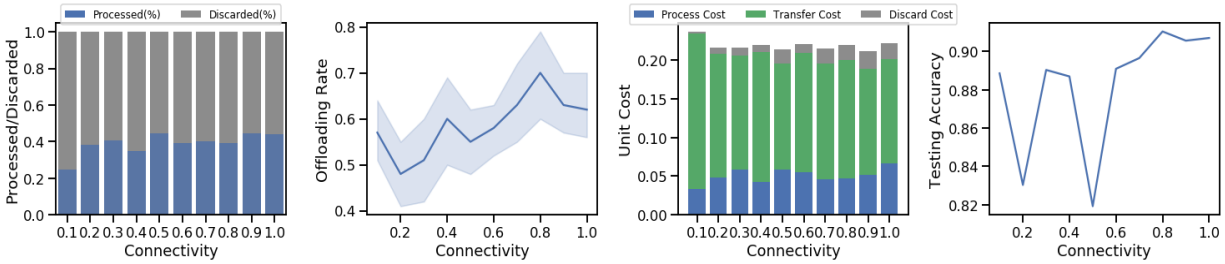


Fig. 7: Impact of the network connectivity ρ on data movement and costs in network-aware learning. Overall, we see that the costs are reasonably robust to ρ, with nodes offloading less frequently as ρ decreases. The learning accuracy, by contrast, tends to improve with ρ.

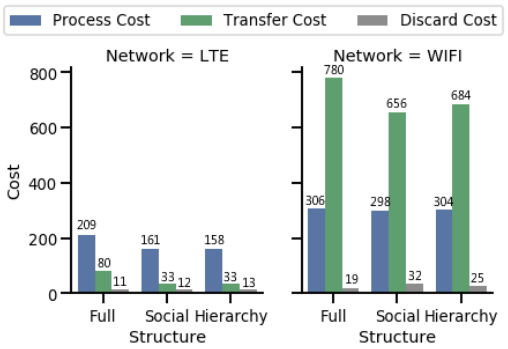


Fig. 8: Cost components for social, hierarchical, and fully connected topologies running network-aware learning on (a) LTE and (b) WiFi network media. Transmission costs dominate for each topology in the case of WiFi, while for LTE the processing costs are the largest.

Varying network connectivity ρ. In Figure 7, we plot the (a) fraction of processed data, (b) offloading rate, (c) costs, and (d) learning accuracy as the network connectivity changes. We see that our network-aware learning methodology is reasonably robust to ρ in terms of cost (for $\rho > 0.1$): when connectivity drops, there are less low-cost opportunities for offloading, so nodes do not transfer as much data. The learning accuracy, by contrast, tends to benefit from higher connectivity (for $\rho > 0.5$), since low cost nodes can process more of the data, similar to setting (B) in Table III. Interestingly, while the percentage of data discarded increases slightly with lower ρ, the discard cost has the opposite trend: when nodes do not have the option of low cost transfers, their $G_i(t)$ become larger for smaller t, causing the discard cost to drop more rapidly as in Lemma 1.

D. Effect of Fog Topology

Finally, we evaluate our network-aware learning methodology on different fog computing topologies. We consider three different graph structures: hierarchical and social network topologies as studied in Section IV-B, and a fully connected topology for completeness. The social network is generated as a Watts-Strogatz small world graph [29] 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. We use the cost parameters collected from our Raspberry Pi testbed, which provides two different wireless network media: LTE and WiFi.

The resulting costs are shown in Figure 8. For LTE, we see that processing dominates the cost distribution for all three topologies, while for WiFi, the transfer costs are the largest: WiFi has less interference mitigation techniques than cellular, so in the presence of several devices we expect its links to exhibit larger delays. This is also likely the reason why the total cost is substantially higher under WiFi, since devices have less lower cost options for offloading. This point is further consistent with the fact that in the case of LTE, the social and hierarchical topologies exhibit virtually the same costs: despite guaranteed connections to higher powered nodes up the hierarchy, the social network likely contains low cost links to these nodes anyway. In the case of WiFi, by contrast, the hierarchical topology has a noticeably higher offloading cost, and lower discarding cost, than the social topology, since the transmissions to high-power nodes occurs over high cost links.

VI. Conclusion and Future Work

To the best of our knowledge, this paper is the first work to consider distributing ML training tasks over devices in a fog computing network. We develop a framework to optimize the distribution of training tasks, taking into account both physical computing and communication costs and the error achieved by the models at each device. We derive new error bounds when devices can transfer their local data processing to each other, and theoretically bound the impact of these transfers on the cost and accuracy of the model training. Through experimentation with a popular machine learning task, we show that our network-aware scheme significantly reduces the cost of model training while achieving comparable accuracy.

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 in offloading data processing.

Acknowledgment

We thank the reviewers for their valuable comments. This work was partially supported by NSF CNS-1909306 and the Army Research Office under grant W911NF1910036.

REFERENCES

[1] 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

[2] 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.

[3] 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

[4] 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.

[5] 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.

[6] 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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Technical Report. www.cbrinton.net/fog-infocom-2020-tech.pdf.

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

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

[25] 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.

[26] F. M. F. Wong, Z. Liu, M. Chiang, F. Ming Fai Wong, Z. Liu, and M. Chiang, "On the Efficiency of Social Recommender Networks," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2512–2524, 2016.

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

[28] 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.

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