Adaptive Sparsification for Communication-Efficient Distributed Learning

Faeze Moradi Kalarde
Ben Liang
University of Toronto
Toronto, ON, Canada

Min Dong
Ontario Tech University
Oshawa, ON, Canada

Yahia A. Eldemerdash Ahmed
Ho Ting Cheng
Ericsson Canada
Ottawa, ON, Canada

Abstract

This work addresses the trade-off between convergence and the overall delay in heterogeneous distributed learning systems, where the devices encounter diverse and dynamic communication conditions. We propose to apply adaptive sparsification across the devices and over iterations, formulating an optimization problem to minimize the overall delay while ensuring a specified level of convergence. The resultant stochastic optimization problem cannot be handled by conventional Lyapunov optimization techniques due to the dependency of the per-iteration objective function on the previous iterations. To overcome this challenge, we propose AdaSparse, an online algorithm with a novel per-slot problem that can be solved optimally by searching over a finite discrete space. We further introduce a low-complexity approximation of AdaSparse, termed LC-AdaSparse, which features linear computational complexity and diminishing approximation error. We show that AdaSparse offers strong performance guarantees, simultaneously achieving sub-linear dynamic regret in terms of delay and the optimal rate in terms of convergence. Numerical experiments on classification tasks using standard datasets and various models demonstrate that our approach effectively reduces the communication delay compared with existing benchmarks, to achieve the same levels of learning accuracy.

CCS Concepts

• **Computing methodologies** → **Machine learning**; **Distributed computing methodologies**.

Keywords

Distributed Learning, Compression, Sparsification, Delay

ACM Reference Format:
Faeze Moradi Kalarde, Ben Liang, Min Dong, Yahia A. Eldemerdash Ahmed, and Ho Ting Cheng. 2025. Adaptive Sparsification for Communication-Efficient Distributed Learning. In *The Twenty-sixth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc '25), October 27–30, 2025, Houston, TX, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3704413.3764446

1 Introduction

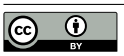
Distributed learning techniques such as distributed stochastic gradient descent (DSGD) [27] rely on frequent parameter exchange between the server and the devices. With the advent of massive deep neural networks (DNNs) comprising millions to billions of parameters, distributed learning imposes a significant communication burden on the network. To reduce the volume of data exchanged, sparsification—a prominent compression technique—has been proposed, wherein each device transmits only a selected subset of gradient vector entries to the server [1, 3, 19, 20]. This approach can significantly speed up the training process by lowering per-iteration communication delays. However, its lossy nature can negatively impact training performance. Thus, an effective sparsification strategy must balance the trade-off between convergence and the overall communication delay.

Furthermore, in a heterogeneous system, the devices experience diverse and dynamic environments, e.g., edge devices in a federated learning (FL) setting may face different wireless channel conditions. Assigning a uniform sparsification level to all devices is inefficient since the devices with slower communication capabilities would cause greater delays, potentially increasing the overall delay significantly. Therefore, the compression design should take the system heterogeneity into account. In addition, during model training, both the changing communication environment and the gradient convergence introduce variation in the optimal sparsification level over time. This leads to a challenging online optimization problem to achieve dynamic adaptation both *across the devices* and *throughout the model training iterations*.

While various studies aim to improve the convergence [5, 10, 14, 18, 23] of sparsification methods, the explicit trade-off between convergence and the overall communication delay remains largely under-explored, particularly in dynamic and heterogeneous environments where the devices' communication conditions vary during training (see further details in Section 2). In this work, we aim to bridge this gap by addressing the aforementioned trade-off. We consider sparsification in a distributed learning system where the per-coordinate transmission delay varies across devices and iterations. Our approach is to minimize the overall communication delay required to achieve a certain level of convergence for the loss function, by adaptively designing individualized and time-varying sparsification levels for each device. To the best of our knowledge, none of the existing schemes for sparsification directly addresses the trade-off between convergence and delay.

This paper makes the following contributions:

- We propose an adaptive sparsification method, referred to as *AdaSparse*, to enhance communication efficiency in DSGD. We formulate an optimization problem with a long-term objective function that accounts for the system's overall delay over an arbitrary number of training iterations while imposing a constraint to ensure a specified level of convergence



MobiHoc '25, Houston, TX, USA

ACM ISBN 979-8-4007-1353-8/2025/10
https://doi.org/10.1145/3704413.3764446

for the loss function. We further derive a sufficient condition for convergence and reformulate the problem using this condition as a surrogate convergence constraint.
- The reformulated problem involves both a long-term objective function and a long-term constraint, rendering it challenging to solve. Furthermore, we observe an implicit dependency of the per-iteration objective function on optimization variables from previous iterations, so that standard Lyapunov optimization techniques cannot be applied. Nevertheless, using a virtual queue approach we develop an online algorithm for designing the sparsification levels. We further propose LC-AdaSparse, a low-complexity variant of AdaSparse with linear computational complexity and diminishing approximation gap.
- We analyze AdaSparse and establish an upper bound for its dynamic regret, defined as the difference between the delay achieved by the proposed algorithm and that of an algorithm solving the per-iteration problem optimally. We demonstrate that, under the assumption of bounded per-coordinate delay, AdaSparse achieves $O(T^{\frac{3-\mu}{2}})$ dynamic regret, where T is the number of training iterations and μ is a tunable parameter. Furthermore, we show that when $\mu \leq 2$ AdaSparse recovers the $O(\frac{1}{\sqrt{T}})$ convergence rate for standard DSGD without sparsification. Therefore, for $1 < \mu \leq 2$, AdaSparse achieves both sub-linear dynamic regret in terms of delay and the optimal rate in terms of convergence.
- We conduct numerical experiments on distributed learning with common classification tasks. Our results show that both AdaSparse and LC-AdaSparse outperform several existing alternatives, substantially reducing the overall communication delay to achieve the same level of accuracy.

2 Related Work

2.1 Fixed Sparsification

In the standard sparsification approach, only a subset of the entries from the local stochastic gradient, along with their indices, are transmitted. Well-known vanilla sparsification approaches include Top-k [1], which selects the k entries with the highest magnitudes for transmission, and the threshold-v method [19], which chooses entries whose absolute values exceed a threshold v.

These vanilla sparsification approaches are usually combined with other techniques to enhance convergence. For example, the error-feedback mechanism stores the error introduced by sparsification, which is then added to the gradient in the next iteration to improve convergence [3, 20]. As another example, Deep Gradient Compression (DGC) [14] enhances training performance by incorporating momentum correction, local gradient clipping, momentum factor masking, and warm-up training. Several approaches aim to further improve the convergence of the Top-k method by approximating the global Top-k entries, which are the k entries with the largest magnitudes in the sum of all local gradients [5, 10, 18]. All of these sparsification approaches apply a fixed sparsification scheme (e.g., fixed k in Top-k) across devices and iterations, ignoring the heterogeneous and dynamic conditions of the network.

2.2 Adaptive Sparsification

Several works have proposed adaptive schemes that adjust the sparsification levels over the training iterations [8, 15, 21, 23]. An unbiased sparse coding method was proposed in [23] to minimize the expected number of entries sent by each device while constraining the variance of the sparsified gradient to improve the convergence rate. To minimize the overall training time, [8] adaptively adjusts k in Top-k across iterations. The work in [15] jointly optimizes the sparsification and quantization levels of devices to minimize their total energy consumption. The work in [21] aims to adjust k in Top-k across iterations to minimize a per-iteration cost that reflects both training performance and communication cost. However, the design objectives in these works differ from ours, as they do not explicitly explore the trade-off between convergence and the overall communication delay, a critical aspect in dynamic and heterogeneous environments where the devices' communication conditions vary during training. In comparison, we propose an online optimization design to address this trade-off.

2.3 Other Compression Methods

Beyond sparsification, general compression techniques are also used to alleviate the communication bottleneck in distributed learning. For example, post-sparsification filtering was proposed in [22], and the importance of different neural network layers was considered in [26]. Furthermore, quantization reduces the number of bits to represent each entry of the gradient vector (e.g., [2]). The low-rank approach decomposes the gradient vector into low-rank matrices before transmission (e.g., [25]). Our work is orthogonal to these compression techniques, and often can be combined with them in the same system to further improve communication efficiency (e.g., [4, 7, 24]). Such hybrid solutions are outside the scope of this work.

3 Preliminaries

3.1 Distributed Learning System Model

We consider a distributed learning system consisting of M devices. Each device $m \in \{1, \ldots, M\}$ is associated with a local loss function $f_m(\mathbf{w})$, where $\mathbf{w} \in \mathbb{R}^n$ denotes the decision vector. The objective is to minimize, in a distributed manner, the global loss function, defined as

$$f(\mathbf{w}) = \frac{1}{M} \sum_{m=1}^{M} f_m(\mathbf{w}). \tag{1}$$

The optimal decision vector that minimizes the global loss function is denoted by $\mathbf{w}^\star$.

A prominent example of the above problem is federated learning, where the devices collaboratively train a machine learning model $\mathbf{w}$ on distributed data across all devices, such that it accurately predicts the true labels of feature vectors for all devices. In this setting, the local loss function of device m is defined as

$$f_m(\mathbf{w}) \triangleq \mathbb{E}_{x_m \sim p_m}\, l_m(\mathbf{w}; x_m), \tag{2}$$

where the expectation is taken over data points x_m sampled from the local data distribution of device m, denoted by p_m, and $l_m(\cdot)$ denotes the sample-wise loss function of device m.

DSGD is a standard approach to solve this problem. At each iteration, each device computes a stochastic gradient using a mini-batch of its local data as an estimate of the local gradient. These local gradients are then sent to a central server for aggregation, to update the global model parameters. This process is repeated iteratively until convergence. While DSGD effectively leverages the computational power of multiple devices, it can lead to substantial communication overhead, especially since the number of model parameters, n, is usually large in machine learning applications.

3.2 Communication Model

Each device applies a sparsification method to its computed stochastic gradient before transmission to reduce the communication overhead. The sparse signal transmitted by device m in iteration t is denoted by $\tilde{\mathbf{g}}_{m,t} \in \mathbb{R}^n$. The number of bits required to transmit the values of non-zero entries of $\tilde{\mathbf{g}}_{m,t}$ is proportional to their count, as each non-zero entry must be encoded using a specific number of bits. Additionally, the index of each non-zero entry can be encoded using $\log_2(n)$ bits. Thus, the delay incurred by each device for transmitting its sparse vector is proportional to the number of non-zero entries, $\|\tilde{\mathbf{g}}_{m,t}\|_0$, where $\|\cdot\|_0$ represents the ℓ_0 norm.

We define $d_{m,t}$ as the delay incurred by device m in iteration t for transmitting a single coordinate of the gradient along with its index, computed as the ratio between the number of bits required to transmit each entry and the device data rate. Therefore, the total delay caused by device m is given by

$$\tau_{m,t} \triangleq d_{m,t}\|\tilde{\mathbf{g}}_{m,t}\|_0. \tag{3}$$

As explained in Section 1, an important feature of this work is that $d_{m,t}$ can vary among the devices and across training iterations and is unknown prior to the start of the t-th iteration.[1]

We consider a shared communication bandwidth among the devices, which is a prominent characteristic of wireless systems and it is also applicable to some wired systems with shared input/output ports. Without loss of generality, we assume a time-division multiple access scheme. Thus, the total communication delay for the system in iteration t is the summation of delays across all devices, expressed as

$$\tau_t \triangleq \sum_{m=1}^{M} \tau_{m,t} = \sum_{m=1}^{M} d_{m,t}\|\tilde{\mathbf{g}}_{m,t}\|_0. \tag{4}$$

We note that the above delay model is applicable also in the case of frequency division multiple access, by standard conversion between time and frequency.

4 Framework for Distributed Learning with Adaptive Sparsification

Distributed learning that incorporates threshold-based sparsification with a *fixed* threshold across devices and iterations was studied in [19]. Here, we describe an adaptive version of the threshold-based sparsification method, where the sparsification threshold can vary across devices and iterations of the algorithm to account for device heterogeneity and the varying conditions during training. Later, in Section 5, we present a strategy for adaptively adjusting the threshold values to minimize the overall communication delay.

The devices execute the DSGD algorithm with individualized and time-varying sparsification to minimize the global loss function. Specifically, during iteration t of the algorithm, the following steps are performed:

(1) **Stochastic gradient computation:** Each device indexed by m, computes an unbiased stochastic gradient on a minibatch of data sampled from its local training dataset:

$$\mathbf{g}_{m,t} = \nabla f_m(\mathbf{w}_t) + \mathbf{z}_{m,t}, \ \ \mathbb{E}[\mathbf{z}_{m,t}|\mathbf{w}_t] = 0, \tag{5}$$

where $\mathbf{z}_{m,t}$ is a zero-mean noise that arises from stochastic sampling.

(2) **Error feedback:** Each device scales a stored error vector $\mathbf{e}_{m,t}$ by the learning rate γ_t, and aggregates it with the stochastic gradient as follows:

$$\hat{\mathbf{g}}_{m,t} = \mathbf{g}_{m,t} + \frac{\mathbf{e}_{m,t}}{\gamma_t}. \tag{6}$$

Initially, the error vectors are set to zero, i.e., $\mathbf{e}_{m,0} = \mathbf{0}, \forall m$.

(3) **Sparsification:** Each device applies sparsification on $\hat{\mathbf{g}}_{m,t}$:

$$\tilde{\mathbf{g}}_{m,t} = \mathcal{S}_{\lambda_{m,t}}(\hat{\mathbf{g}}_{m,t}). \tag{7}$$

where $\mathcal{S}_{\lambda_{m,t}} : \mathbb{R}^n \to \mathbb{R}^n$ is the sparsification operation, i.e.,

$$\tilde{g}^i_{m,t} = \begin{cases} \hat{g}^i_{m,t}, & \text{if } |\hat{g}^i_{m,t}| > \lambda_{m,t}, \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

where $\hat{g}^i_{m,t}$ and $\tilde{g}^i_{m,t}$ represent the i-th entry of $\hat{\mathbf{g}}_{m,t}$ and $\tilde{\mathbf{g}}_{m,t}$ respectively.

(4) **Error update:** Each device updates its error vector based on the difference between $\hat{\mathbf{g}}_{m,t}$ and $\tilde{\mathbf{g}}_{m,t}$ as

$$\mathbf{e}_{m,t+1} = \gamma_t(\hat{\mathbf{g}}_{m,t} - \tilde{\mathbf{g}}_{m,t}). \tag{9}$$

(5) **Averaging:** The server computes the average of received sparse vectors $\tilde{\mathbf{g}}_{m,t}$, across devices as

$$\Delta_t = \frac{1}{M}\sum_{m=1}^{M} \tilde{\mathbf{g}}_{m,t}. \tag{10}$$

(6) **Model update and broadcast:** The severe updates the model based on the average of sparse vectors, i.e.,

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t \Delta_t, \tag{11}$$

and broadcasts the updated model to all devices.

This procedure is summarized in Algorithm 1.

5 Adaptive Sparsification Design

Although aggressive sparsification, i.e., setting the sparsification thresholds $\{\lambda_{m,t}\}$ to high values, results in low delay in each iteration, it degrades convergence by introducing significant deviation from the true gradient during model updates. Conversely, lighter sparsification can improve convergence but leads to higher per-iteration delay. In this section, we present an online optimization solution to design the sparsification thresholds $\{\lambda_{m,t}\}$ to address this trade-off.

[1] For example, in wireless systems, the dynamic nature of channel conditions results in uncertain delays across iterations.

Algorithm 1 DSGD with Adaptive Sparsification

Inputs: $\mathbf{w}_0$, $\{\gamma_t\}$, $\{\lambda_{m,t}\}$
Output: $\{\mathbf{w}_t\}$
Initialization: $\mathbf{e}_{m,0} = \mathbf{0}, \forall m$.
for $t \in \{0, ..., T-1\}$ **do**
  Each device m **in parallel do:**
    Computation of stochastic gradient $\mathbf{g}_{m,t}$.
    $\hat{\mathbf{g}}_{m,t} = \mathbf{g}_{m,t} + \frac{\mathbf{e}_{m,t}}{\gamma_t}$ ▷ Error feedback
    $\tilde{\mathbf{g}}_{m,t} = \mathcal{S}_{\lambda_{m,t}}(\hat{\mathbf{g}}_{m,t})$ ▷ Sparsification
    $\mathbf{e}_{m,t+1} = \gamma_t(\hat{\mathbf{g}}_{m,t} - \tilde{\mathbf{g}}_{m,t})$ ▷ Error update
    Send $\tilde{\mathbf{g}}_{m,t}$ to the server.
  The server does:
    $\Delta_t = \frac{1}{M}\sum_{m=1}^{M} \tilde{\mathbf{g}}_{m,t}$. ▷ Averaging
    $\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t \Delta_t$ ▷ Model update and broadcast
end for

5.1 Adaptive Optimization of Sparsification Threshold

We formulate an optimization problem, where the objective is to minimize the total delay after a predetermined number of training iterations, T, while ensuring a certain level of convergence of the model to a stationary point of the global loss function:

$$\min_{\{\lambda_{m,t}\}} \quad \sum_{t=0}^{T-1}\sum_{m=1}^{M} \tau_{m,t} \tag{12a}$$

$$\text{s.t.} \quad \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\|\nabla f(\mathbf{w}_t)\|^2 \le \delta, \tag{12b}$$

where the $\mathbb{E}[\cdot]$ is on the randomness of the stochastic gradients, i.e., $\{\mathbf{z}_{m,t}\}$. In this formulation, the objective function (12a) represents the total system delay over T iterations. The constraint (12b) ensures that the system achieves δ-convergence to a stationary point of the global loss function $f(\mathbf{w})$. We note that bounding the left-hand side (LHS) of (12b) implies a bound on $\min_{0\le t\le T-1}\mathbb{E}\|\nabla f(\mathbf{w}_t)\|^2$, which guarantees that at least a model among $\{\mathbf{w}_t\}$ during the training process will be sufficiently close to a stationary point if δ is chosen to be small enough.

Solving (12) presents significant challenges because the constraint involves the gradient of the global loss function, which is not an explicit function of the optimization variables. Also, the global loss function is typically not quantifiable, as it depends on the local data distributions $\{p_m\}$, which are unknown to the devices. To proceed, we analyze the convergence of DSGD with sparsification and substitute (12b) with a more manageable surrogate constraint.

5.2 Problem Reformulation via Training Convergence Analysis

5.2.1 Convergence Analysis. We consider the following four assumptions on the loss function, which are common in the literature of distributed learning [3, 19]:

A1. Smoothness: The local loss function of each device m is L-smooth, i.e., $\forall \mathbf{w}, \mathbf{w}' \in \mathbb{R}^n$,

$$f_m(\mathbf{w}) \le f_m(\mathbf{w}') + \left\langle \nabla f_m(\mathbf{w}'), \mathbf{w} - \mathbf{w}' \right\rangle + \frac{L}{2}\|\mathbf{w} - \mathbf{w}'\|^2. \tag{13}$$

A2. Global minimum: There exists $\mathbf{w}^\star \in \mathbb{R}^n$ such that

$$f(\mathbf{w}^\star) = f^\star \le f(\mathbf{w}),\ \forall \mathbf{w} \in \mathbb{R}^n. \tag{14}$$

A3. Bounded noise: For every stochastic noise $\mathbf{z}_{m,t}$, there exist $A, \sigma^2 \ge 0$, such that

$$\mathbb{E}[\|\mathbf{z}_{m,t}\|^2 | \mathbf{w}_t] \le A\|\nabla f_m(\mathbf{w}_t)\|^2 + \sigma^2,\ \forall \mathbf{w}_t \in \mathbb{R}^n. \tag{15}$$

A4. Bounded similarity: The variance of gradient among devices is bounded, i.e., there exist constants, $C, \zeta \ge 0$ such that, $\forall \mathbf{w} \in \mathbb{R}^n$,

$$\frac{1}{M}\sum_{m=1}^{M} \|\nabla f_m(\mathbf{w}) - \nabla f(\mathbf{w})\|^2 \le C\|\nabla f(\mathbf{w})\|^2 + \zeta^2. \tag{16}$$

In the following proposition, we provide a convergence bound for adaptive sparsification where the threshold values vary across iterations and devices.

Proposition 1. *Assuming that conditions **A1**-**A4** hold and the learning rate is constant and bounded, i.e., $\gamma_t = \gamma$ such that $\gamma \le \frac{M}{2L(A(C+1)+M)}$, the sequence $\{\mathbf{w}_t\}$ generated by Algorithm 1 satisfies the following:*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\|\nabla f(\mathbf{w}_t)\|^2 \le \frac{4}{\gamma T}\big(f(\mathbf{w}_0) - f^\star\big) + \frac{2\gamma L}{M}\big(A\zeta^2 + \sigma^2\big) + \frac{2nL^2\gamma^2}{MT}\sum_{t=0}^{T-1}\sum_{m=1}^{M}\lambda_{m,t}^2. \tag{17}$$

Proof. Define $\tilde{\mathbf{w}}_t \triangleq \mathbf{w}_t - \frac{1}{M}\sum_{m=1}^{M}\mathbf{e}_{m,t}$. We have

$$\begin{aligned}
\mathbb{E}[f(\tilde{\mathbf{w}}_{t+1})] &\overset{(a)}{\le} \mathbb{E}[f(\tilde{\mathbf{w}}_t)] - \frac{\gamma}{4}\mathbb{E}\|\nabla f(\mathbf{w}_t)\|^2 + \frac{\gamma^2 L(A\zeta^2+\sigma^2)}{2M} + \frac{\gamma L^2}{2M}\sum_{m=1}^{M}\mathbb{E}\|\mathbf{e}_{m,t}\|^2 \\
&\overset{(b)}{\le} \mathbb{E}[f(\tilde{\mathbf{w}}_t)] - \frac{\gamma}{4}\mathbb{E}\|\nabla f(\mathbf{w}_t)\|^2 + \frac{\gamma^2 L(A\zeta^2+\sigma^2)}{2M} + \frac{\gamma^3 L^2 n}{2M}\sum_{m=1}^{M}\lambda_{m,t}^2,
\end{aligned} \tag{18}$$

where (a) is due to [19, Lemma 13], and (b) follows from the bound on the error norm as $\|\mathbf{e}_{m,t}\|^2 \le n\gamma^2\lambda_{m,t}^2, \forall m, \forall t$, which holds because the entries of the error vector $\mathbf{e}_{m,t}$ are always less than or equal to $\gamma\lambda_{m,t}$. Summing both sides of (18) from iteration 0 to $T-1$, rearranging the terms, and dividing by γT completes the proof. □

Note that the first two terms of the upper bound in (17) are constant since they do not depend on the decision variables $\{\lambda_{m,t}\}$. For simplicity, we define these terms as

$$\phi \triangleq \frac{4}{\gamma T}(f(\mathbf{w}_0) - f^\star) + \frac{2\gamma L(A\zeta^2+\sigma^2)}{M}. \tag{19}$$

5.2.2 Problem Reformulation. We leverage the result in Proposition 1 to replace the LHS of (12b) with its upper bound. To ensure that $\frac{1}{T}\sum_{t=0}^{T-1}\|\nabla f(\mathbf{w}_t)\|^2$ is bounded by δ, it suffices to bound the right-hand side (RHS) of (17) by the same amount. Since the first two terms of RHS of (17) are constant, restricting it by δ implies a bound on the third term $\frac{1}{MT}\sum_{t=0}^{T-1}\sum_{m=1}^{M}\lambda_{m,t}^2$ by ϵ, where $\epsilon = \frac{\delta-\phi}{2nL^2\gamma^2}$.[2] Therefore, we reformulate problem (12) as

$$\min_{\{\lambda_{m,t}\}} \quad \sum_{t=0}^{T-1}\sum_{m=1}^{M}\tau_{m,t} \tag{20a}$$

$$\text{s.t.} \quad \frac{1}{MT}\sum_{t=1}^{T-1}\sum_{m=1}^{M}\lambda_{m,t}^2 \le \epsilon, \tag{20b}$$

where ϵ replaces δ as the hyperparameter to tune the trade-off between convergence and the overall delay.

The above problem is still difficult to handle for the following reasons: *(i)* Tackling the long-term objective and constraints is challenging because the per-coordinate delay values, $\{d_{m,t}; 1 \le m \le M\}$, are unknown prior to the start of the t-th iteration and only become available at its onset. *(ii)* $\tau_{m,t}$ is a function of $\hat{\mathbf{g}}_{m,t}$, which cannot be determined before the start of the t-th iteration. *(iii)* Global information about $\{\hat{\mathbf{g}}_{m,t}\}$ is unavailable across devices, as only their sparse versions, $\{\tilde{\mathbf{g}}_{m,t}\}$, are transmitted by the devices.

We note that even though problem (20) has a standard long-term constraint, the conventional Lyapunov optimization framework [17] is not an applicable solution. In Lyapunov optimization, the per-iteration objective function and constraint in iteration t should depend only on the current decision variables, i.e., $\{\lambda_{m,t}; 1 \le m \le M\}$, and the current randomness of the system, i.e., $\{d_{m,t}; 1 \le m \le M\}$. In contrast, in problem (20), the per-iteration objective function $\sum_{m=1}^{M}\tau_{m,t}$ depends on the decision variable of current and *previous* iterations, i.e., $\{\lambda_{m,c}; 1 \le m \le M, c \le t\}$. This is due to the dependence of $\tau_{m,t}$ on $\hat{\mathbf{g}}_{m,t}$ based on (3) and (7), which itself depends on the gradients computed at $\mathbf{w}_t$, i.e., $\mathbf{g}_{m,t}$ based on (6). In turn, $\mathbf{w}_t$ is updated in the previous iteration, making it dependent on the decision variables from all earlier iterations.

Instead, we propose a new algorithm to solve the problem in an online manner and provide bounds for both the constraint violation and the dynamic regret.

5.3 AdaSparse Algorithm

To keep track of the constraint violation, we define a virtual queue for each device m denoted by q_t^m with $q_0^m = 0, \forall m$. In each iteration t, each device updates its virtual queue as

$$q_{t+1}^m = \max\left\{q_t^m + \lambda_{m,t}^2 - \epsilon, 0\right\}, \forall m. \tag{21}$$

In departure from Lyapunov optimization, which is *not* applicable to our problem as explained above, we consider a different form of the per-slot optimization as follows. In iteration t, each device m solves an optimization problem to design its sparsification level as

$$\min_{\lambda_{m,t}} \quad V\tau_{m,t} + q_t^m\lambda_{m,t}^2 + \frac{1}{2}\lambda_{m,t}^4, \tag{22}$$

where $V \in \mathbb{R}^+$ is a predefined constant. Note that $\tau_{m,t}$ depends on $\lambda_{m,t}$ via (3) and (7).

Although (22) is a single-variable unconstrained problem, it is challenging to solve because it is non-convex and non-differentiable, as $\tau_{m,t}$ involves the ℓ_0 norm, which is non-differentiable. However, the following proposition simplifies solving it by providing a finite set of candidate optimal solutions.

Proposition 2. *The optimal value of $\lambda_{m,t}$ for* (22) *belongs to the following finite set:*

$$\mathcal{J} = \left\{0, |\hat{g}_{m,t}^{(1)}|, |\hat{g}_{m,t}^{(2)}|, ..., |\hat{g}_{m,t}^{(n)}|\right\}, \tag{23}$$

where $|\hat{g}_{m,t}^{(i)}|$ is the i-th largest entry of $\hat{\mathbf{g}}_{m,t}$ in terms of magnitude, i.e.,$|\hat{g}_{m,t}^{(1)}| \le |\hat{g}_{m,t}^{(2)}| \le ... \le |\hat{g}_{m,t}^{(n)}|$.

Proof. The proof is by contradiction. Assume that an optimal solution for (22), denoted by $\lambda_{m,t}^\star$, does not belong to $\mathcal{J}$. Then, one of the following three cases must hold: *(i)* $0 < \lambda_{m,t}^\star < |\hat{g}_{m,t}^{(1)}|$, *(ii)* $|\hat{g}_{m,t}^{(j)}| < \lambda_{m,t}^\star < |\hat{g}_{m,t}^{(j+1)}|$ for some $1 \le j \le n-1$, or *(iii)* $\hat{g}_{m,t}^{(n)} < \lambda_{m,t}^\star$. In the following, we show that $\lambda_{m,t}^\star$ cannot be an optimal solution in any of these cases.

Suppose case *(ii)* holds. Then, we can introduce a new variable $\lambda_{m,t}^\dagger = |\hat{g}_{m,t}^{(j)}|$, which achieves a lower value of the objective function in (22). This is because the value of the first term in the objective function, $\tau_{m,t}$, remains the same under both $\lambda_{m,t}^\dagger$ and $\lambda_{m,t}^\star$, as it only depends the number of elements in $\hat{\mathbf{g}}_{m,t}$ greater than the sparsification threshold, which in both cases is $n-j$. However, the last two terms of the objective function are increasing functions of $\lambda_{m,t}$, and since $\lambda_{m,t}^\dagger$ is smaller than $\lambda_{m,t}^\star$, the overall value of the objective function is smaller under $\lambda_{m,t}^\dagger$. This contradicts the definition of $\lambda_{m,t}^\star$ as an optimal solution for (22).

By similar reasoning, we can show that the optimal solution cannot satisfy cases *(i)* and *(iii)* either. The details are omitted due to the page limit. Hence, the optimal solution must lie within the discrete set $\mathcal{J}$. □

Proposition 2 identifies the candidate optimal solutions for (22). By evaluating the objective function for each value within $\mathcal{J}$ and selecting the one that yields the minimum value, we can determine the optimal solution. The computational complexity of this procedure is $O(n \log n)$, as it involves sorting $n+1$ values.

Substituting the optimal $\{\lambda_{m,t}\}$ into Algorithm 1, we obtain the proposed AdaSparse algorithm.

5.4 Low-Complexity AdaSparse

Noting that n can be large in many machine learning applications, to further reduce the complexity of AdaSparse, we propose a low-complexity version called *Low-Complexity AdaSparse* (*LC-AdaSparse*). The most computationally demanding part of AdaSparse is solving (22) optimally, which requires sorting the n candidate values in the set $\mathcal{J}$. In practice, however, an approximate solution to (22) can significantly lower the computational complexity. Specifically, for the entries of the gradient vector in the set $\mathcal{J}$ whose absolute values are close together, the last two terms in (22) remain relatively unchanged. Thus, the first term can be minimized by considering only the largest element among these values.

[2]Note that ϕ decreases as T increases when $\gamma \propto T^\psi$, for $-1 < \psi < 0$. Thus, ϕ can be made arbitrarily small by setting T large, to avoid $\delta - \phi$ becoming negative.

Based on this observation, we divide the range of values in $\mathcal{J}$, i.e., $\left[0, |\hat{g}_{m,t}^{(n)}|\right]$, into B intervals, each of length $l \triangleq \frac{|\hat{g}_{m,t}^{(n)}|}{B}$. Next, we count the number of elements in $\mathcal{J}$ that fall within each i-th interval, defined as $\big((i-1)l, il\big]$. This yields a count vector $\mathbf{v}_c \in \mathbb{R}^B$, where each entry represents the number of elements in $\mathcal{J}$ within the corresponding interval. We then consider the upper bound of each interval, il for $1 \le i \le B$, as a candidate threshold for all the elements inside that interval. Let $\tilde{\mathcal{J}} = \big\{il \mid 1 \le i \le B\big\}$. Using the counts in $\mathbf{v}_c$, we evaluate the objective function of (22) at the values in $\tilde{\mathcal{J}}$ and select the one that minimizes the objective function.

LC-AdaSparse avoids sorting n candidate values to achieve a lower computational complexity than AdaSparse. Computing the count vector requires $O(n)$ operations, achieved by dividing each element of $\mathcal{J}$ by the interval length l and indexing the corresponding interval. Given $\mathbf{v}_c$, evaluating the objective function at each value in $\tilde{\mathcal{J}}$ and finding the minimum requires an additional $O(B)$ operations. Thus, the overall computational complexity is $O(n+B)$.

Since B does not depend on n, the complexity of LC-AdaSparse is linear in n, matching the order of complexity of other conventional sparsification methods. For example, the complexity of Top-k sparsification is $O(n \log k)$, while both threshold-based sparsification and the VarReduced method in [23] have a complexity of $O(n)$.

Next, we investigate the approximation gap for LC-AdaSparse. Let $O^\star$ be the minimum of the objective in (22), and $\tilde{O}$ be the objective value achieved by LC-AdaSparse.

Proposition 3. *The approximation gap of LC-AdaSparse is upper-bounded by*

$$\tilde{O} - O^\star \le l^2 q_t^m (2B-1) + \frac{l^4}{2}\left(4B^3 - 6B^2 + 4B - 1\right). \tag{24}$$

Proof. Suppose the optimal solution $\lambda^\star$ to problem (22) lies in the j-th interval, i.e., $(j-1)l < \lambda^\star \le jl$. Since LC-AdaSparse selects the interval upper bound that minimizes the objective function across all intervals, $\tilde{O}$ is less than or equal to the value of the objective function at $\lambda_{m,t} = jl$. Thus, the difference between $\tilde{O}$ and $O^\star$ is upper-bounded as $\tilde{O} - O^\star \le V(\tau_{m,t}^{(j)} - \tau_{m,t}^\star) + q_t^m\left(j^2l^2 - (\lambda^\star)^2\right) + \frac{1}{2}\left(j^4l^4 - (\lambda^\star)^4\right)$, where $\tau_{m,t}^{(j)}$ is the delay achieved when $\lambda_{m,t} = jl$, and $\tau_{m,t}^\star$ is the delay under $\lambda^\star$.

Since $\lambda^\star \le jl$, $\tau_{m,t}^{(j)} \le \tau_{m,t}^\star$, and we have

$$\begin{aligned}
\tilde{O} - O^\star &\le q_t^m\left(j^2l^2 - (\lambda^\star)^2\right) + \frac{1}{2}\left(j^4l^4 - (\lambda^\star)^4\right) \\
&\overset{(a)}{\le} q_t^m l^2\left(j^2 - (j-1)^2\right) + \frac{l^4}{2}\left(j^4 - (j-1)^4\right) \\
&\overset{(b)}{=} l^2 q_t^m(2j-1) + \frac{l^4}{2}\left(4j^3 - 6j^2 + 4j - 1\right) \\
&\overset{(c)}{\le} l^2 q_t^m(2B-1) + \frac{l^4}{2}\left(4B^3 - 6B^2 + 4B - 1\right),
\end{aligned} \tag{25}$$

where (a) holds because $(j-1)l < \lambda^\star$, (b) follows from simplifying the terms, and (c) results from substituting B for j, noting monotonicity in j. □

Remark 1. *Substituting $l = \frac{|\hat{g}_{m,t}^{(n)}|}{B}$ into (24) provides a bound for the gap as $\tilde{O} - O^\star \le O\left(\frac{1}{B}\right)$. This demonstrates that, for sufficiently large B, the approximation gap of LC-AdaSparse is negligible. This observation is further substantiated in Section 7.*

6 Theoretical Performance Analysis

Here, we analyze the performance of AdaSparse. We note that although our analysis uses the familiar notion of drift, it is substantially different from the conventional Lyapunov stability analysis, and it leads to novel constraint violation and dynamic regret bounds. To begin, let $\hat{\lambda}_{m,t}$ denote the sparsification threshold of device m in iteration t obtained using AdaSparse, and let $\hat{\tau}_{m,t}$ represent the corresponding delay caused by applying this threshold.

6.1 Upper Bound on Global R-Slot Drift

For any positive integer $R \le T$, define the R-slot drift for each device m as

$$\Delta_R^m(t) \triangleq \frac{1}{2}(q_{t+R}^m)^2 - \frac{1}{2}(q_t^m)^2. \tag{26}$$

Then, the global R-slot drift is the summation of R-slot drifts over all devices:

$$\Delta_R(t) \triangleq \sum_{m=1}^M \Delta_R^m(t) = \frac{1}{2}\sum_{m=1}^M (q_{t+R}^m)^2 - \frac{1}{2}\sum_{m=1}^M (q_t^m)^2. \tag{27}$$

Moreover, based on (27), the global R-slot drift could be equivalently written as the aggregation of the global one-slot drifts:

$$\Delta_R(t) = \sum_{c=t}^{t+R-1} \Delta_1(c). \tag{28}$$

Using (27) and noting that the initial values of the queues are set to zero, we can rewrite the global R-slot drift at time $t = 0$ as $\Delta_R(0) = \frac{1}{2}\sum_{m=1}^M (q_R^m)^2$, which implies an upper bound on q_R^m as

$$q_R^m \le \sqrt{2\Delta_R(0)}, \forall m. \tag{29}$$

The following lemma establishes an upper bound on the one-slot drift for each device.

Lemma 1. *Under AdaSparse, the one-slot drift for device m is upper bounded by*

$$\Delta_1^m(t) \le \hat{\lambda}_{m,t}^2 q_t^m + \frac{1}{2}\hat{\lambda}_{m,t}^4 + \frac{1}{2}\epsilon^2. \tag{30}$$

Proof. Based on the queue update equation in (21), we have $q_{t+1}^m \le \left|q_t^m + \hat{\lambda}_{m,t}^2 - \epsilon\right|$. Squaring both sides of this inequality gives

$$(q_{t+1}^m)^2 \le (q_t^m)^2 + 2q_t^m\left(\hat{\lambda}_{m,t}^2 - \epsilon\right) + \left(\hat{\lambda}_{m,t}^2 - \epsilon\right)^2. \tag{31}$$

Rearranging the terms in (31), we have

$$\Delta_1^m(t) \le \hat{\lambda}_{m,t}^2\left(q_t^m - \epsilon\right) + \frac{1}{2}\hat{\lambda}_{m,t}^4 + \frac{1}{2}\epsilon^2 - \epsilon q_t^m, \tag{32}$$

where further upper bounding by disregarding the negative terms leads to the upper bound given in (30). □

To make an upper bound on the global R-slot drift, $\Delta_R(0)$, we sum both sides of (30) across all devices and over t from 0 to $R-1$. By incorporating (28), we obtain

$$\Delta_R(0) \le \frac{1}{2}\sum_{t=0}^{R-1}\sum_{m=1}^M \hat{\lambda}_{m,t}^4 + \sum_{t=0}^{R-1}\sum_{m=1}^M \hat{\lambda}_{m,t}^2 q_t^m + \frac{MR\epsilon^2}{2}. \tag{33}$$

6.2 Upper Bound on Virtual Queues

Lemma 2. *Under AdaSparse, the virtual queues are upper bounded by*

$$q_t^m \le Q_T^{\max}, 1 \le t \le T, \forall m, \tag{34}$$

where $Q_T^{\max} \triangleq \left(2nV \sum_{t=0}^{T-1} \sum_{m=1}^{M} d_{m,t} + MT\epsilon^2\right)^{\frac{1}{2}}$.

Proof. From (33), we have

$$V \sum_{t=0}^{R-1} \sum_{m=1}^{M} \hat{\tau}_{m,t} + \Delta_R(0) \le V \sum_{t=0}^{R-1} \sum_{m=1}^{M} \hat{\tau}_{m,t} + \frac{1}{2} \sum_{t=0}^{R-1} \sum_{m=1}^{M} \hat{\lambda}_{m,t}^4 + \sum_{t=0}^{R-1} \sum_{m=1}^{M} \hat{\lambda}_{m,t}^2 q_t^m + \frac{MR\epsilon^2}{2}. \tag{35}$$

Since each device under AdaSparse is able to solve (22) optimally, and the RHS of (35) is the summation of the objective function of (22) (up to a constant) over iterations and across devices, AdaSparse achieves the minimum value of the RHS of (35). Thus, plugging in any other values of $\lambda_{m,t}$ and the resultant delay on the RHS of (35), instead of $\hat{\lambda}_{m,t}$, leads to an upper bound for the LHS of (35). In particular, considering $\lambda_{m,t} = 0, \forall m, t$, we obtain

$$V \sum_{t=0}^{R-1} \sum_{m=1}^{M} \hat{\tau}_{m,t} + \Delta_R(0) \le nV \sum_{t=0}^{R-1} \sum_{m=1}^{M} d_{m,t} + \frac{MR\epsilon^2}{2}, \tag{36}$$

since the number of non-zero entries is bounded by the number of entries n. Since both terms on the LHS of (36) are non-negative, we have

$$\Delta_R(0) \le nV \sum_{t=0}^{R-1} \sum_{m=1}^{M} d_{m,t} + \frac{MR\epsilon^2}{2}, \quad 1 \le R \le T. \tag{37}$$

Using (29) together with (37), we can provide an upper bound on the queues as

$$q_R^m \le \left(2nV \sum_{t=0}^{R-1} \sum_{m=1}^{M} d_{m,t} + MR\epsilon^2\right)^{\frac{1}{2}} \overset{(a)}{\le} Q_T^{\max}, \ 1 \le R \le T, \forall m, \tag{38}$$

where (a) follows from $\left(2nV \sum_{t=0}^{R-1} \sum_{m=1}^{M} d_{m,t} + MR\epsilon^2\right)^{\frac{1}{2}} \le Q_T^{\max}$ for $1 \le R \le T$. □

6.3 Constraint Violation Bound

The following proposition provides an upper bound on the amount of violation with respect to the constraint (20b).

Proposition 4. *The constraint violation of AdaSparse is upper bounded as*

$$\frac{1}{MT} \sum_{t=0}^{T-1} \sum_{m=1}^{M} \hat{\lambda}_{m,t}^2 - \epsilon \le \frac{Q_T^{\max}}{T}. \tag{39}$$

Proof. We have

$$\frac{1}{MT} \sum_{t=0}^{T-1} \sum_{m=1}^{M} \hat{\lambda}_{m,t}^2 - \epsilon \overset{(a)}{\le} \frac{1}{MT} \sum_{t=0}^{T-1} \sum_{m=1}^{M} (q_{t+1}^m - q_t^m + \epsilon) - \epsilon \overset{(b)}{=} \frac{1}{MT} \sum_{m=1}^{M} q_T^m \overset{(c)}{\le} \frac{1}{MT} \sum_{m=1}^{M} Q_T^{\max} = \frac{Q_T^{\max}}{T}, \tag{40}$$

where (a) follows from $q_t^m + \hat{\lambda}_{m,t}^2 - \epsilon \le q_{t+1}^m$ based on the queue update equation in (21), (b) is derived after simplifying the terms inside the summation, and (c) is derived using the virtual queue upper bound provided by Lemma 2. □

6.4 Dynamic Regret Bound

Here, we aim to derive an upper bound on the difference between the overall delay under AdaSparse and that of a per-iteration optimal solution.

We first consider a reduced version of (20) by decomposing the long-term objective function and constraint into per-iteration components:

$$\min_{\{\lambda_{m,t}\}} \sum_{m=1}^{M} \tau_{m,t} \quad \text{s.t.} \quad \frac{1}{M} \sum_{m=1}^{M} \lambda_{m,t}^2 \le \epsilon. \tag{41}$$

The solution to (41) is feasible for (20). However, it is impossible to solve (41) in practice. It would require global information about $\{\hat{\mathbf{g}}_{m,t}\}$, which are not obtainable, as the devices only transmit their sparse versions $\{\tilde{\mathbf{g}}_{m,t}\}$. We refer to problem (41) as the per-iteration problem and use the optimal solution to this problem, denoted by $\{\bar{\lambda}_{m,t}\}$, as a benchmark to compare against the solution achieved by AdaSparse for deriving the dynamic regret bound.

The following proposition provides an upper bound on the dynamic regret of AdaSparse which is defined as the gap between the achieved overall delay by AdaSparse and the overall delay by the optimal solution of the per-iteration problem defined in (41), denoted by $\sum_{t=0}^{T-1} \sum_{m=1}^{M} \bar{\tau}_{m,t}$.

Proposition 5. *The dynamic regret of AdaSparse is bounded as*

$$\sum_{t=0}^{T-1} \sum_{m=1}^{M} (\hat{\tau}_{m,t} - \bar{\tau}_{m,t}) \le \frac{\epsilon M^2 T Q_T^{\max}}{V} + \frac{MT\epsilon^2(M^2+1)}{2V}. \tag{42}$$

Proof. We use a similar argument as in the proof of Lemma 2. Using (35) with $R = T$, and considering $\lambda_{m,t} = \bar{\lambda}_{m,t}, \forall m, t$, we obtain

$$\begin{aligned} &V \sum_{t=0}^{T-1} \sum_{m=1}^{M} \hat{\tau}_{m,t} + \Delta_T(0) \\ &\le V \sum_{t=0}^{T-1} \sum_{m=1}^{M} \bar{\tau}_{m,t} + \frac{1}{2} \sum_{t=0}^{T-1} \sum_{m=1}^{M} \bar{\lambda}_{m,t}^4 + \sum_{t=0}^{T-1} \sum_{m=1}^{M} \bar{\lambda}_{m,t}^2 q_t^m + \frac{MT\epsilon^2}{2} \\ &\overset{(a)}{\le} V \sum_{t=0}^{T-1} \sum_{m=1}^{M} \bar{\tau}_{m,t} + \frac{1}{2} \sum_{t=0}^{T-1} \sum_{m=1}^{M} M^2\epsilon^2 + \sum_{t=0}^{T-1} \sum_{m=1}^{M} \epsilon M Q_T^{\max} + \frac{MT\epsilon^2}{2} \\ &\overset{(b)}{=} V \sum_{t=0}^{T-1} \sum_{m=1}^{M} \bar{\tau}_{m,t} + \epsilon M^2 T Q_T^{\max} + \frac{MT\epsilon^2}{2}(M^2+1), \end{aligned} \tag{43}$$

where (a) follows from the fact that $\bar{\lambda}_{m,t}$ belongs to the feasible set of problem (41), and thus $\bar{\lambda}_{m,t}^2 \le \epsilon M, \forall m, \forall t$, and also, $q_t^m \le Q_T^{\max}$ based on Lemma 2; and (b) is obtained by simplifying the summation terms. Dividing both sides of (43) by V and noting that $\Delta_T(0) \ge 0$ completes the proof. □

6.5 Optimality of AdaSparse

Define $D_T \triangleq 2n \sum_{t=0}^{T-1} \sum_{m=1}^{M} d_{m,t}$. The next corollary simplifies the bounds in Propositions 4 and 5.

Corollary 1. *Assuming the per-coordinate delay is bounded above, i.e., $d_{m,t} \le d, \forall m, \forall t$, the constraint violation bound in Proposition 4 and the dynamic regret bound in Proposition 5 reduce to:*

$$\frac{1}{MT}\sum_{t=0}^{T-1}\sum_{m=1}^{M}\hat{\lambda}_{m,t}^2 - \epsilon \le \frac{\sqrt{M(2ndV+\epsilon^2)}}{\sqrt{T}}, \tag{44a}$$

$$\sum_{t=0}^{T-1}\sum_{m=1}^{M}(\hat{\tau}_{m,t}-\bar{\tau}_{m,t}) \le \frac{\epsilon M^2 T}{V}\sqrt{MT\left(2ndV+\epsilon^2\right)} + \frac{MT\epsilon^2}{2V}(M^2+1). \tag{44b}$$

Proof. Replacing D_T with its upper bound $D_T \le 2ndMT$ in Propositions 4 and 5 results in (44a) and (44b) respectively. □

Corollary 2. *For any arbitrary $\mu > 0$, AdaSparse achieves the following trade-off between its constraint violation and dynamic regret:*

$$\frac{1}{MT}\sum_{t=0}^{T-1}\sum_{m=1}^{M}\hat{\lambda}_{m,t}^2 - \epsilon \le O(T^{\frac{\mu-1}{2}}), \tag{45a}$$

$$\sum_{t=0}^{T-1}\sum_{m=1}^{M}(\hat{\tau}_{m,t}-\bar{\tau}_{m,t}) \le O\left(T^{\frac{3-\mu}{2}}\right). \tag{45b}$$

Proof. Setting $V \propto T^{\mu}, \mu > 0$, the bounds in Corollary 1 reduce to (45a) and (45b). □

Finally, the following corollary states that AdaSparse recovers the same asymptotic convergence rate of $O(\frac{1}{\sqrt{T}})$ as the standard DSGD algorithm without sparsification [19].

Corollary 3. *With a constant learning rate $\gamma = O\left(\frac{1}{\sqrt{T}}\right)$ and $V \propto T^{\mu}$ for $\mu \le 2$, AdaSparse achieves an asymptotic rate of $O\left(\frac{1}{\sqrt{T}}\right)$ for model convergence to a stationary point of the global loss function, matching the asymptotic convergence rate of standard DSGD.*

Proof. Utilizing the constraint violation bound (45a) from Corollary 2 and substituting $\gamma = \frac{1}{\sqrt{T}}$ into (17) in Proposition 1, we observe that the first two terms on the RHS are $O(\frac{1}{\sqrt{T}})$, while the third term is also $O(\frac{1}{\sqrt{T}})$ for $\mu \le 2$. This yields $\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(\mathbf{w}_t)\|^2 \le O\left(\frac{1}{\sqrt{T}}\right)$, which completes the proof. □

Remark 2. *We note that the long-term constraint* (20b) *is established solely to ensure training convergence. Therefore, unlike most works on online optimization in the literature, in this work we are less concerned about the constraint violation expressed in* (45a) *than the convergence rate given in Corollary 3. In particular, we observe from* (45b) *that $\mu > 1$ suffices for AdaSparse to have sub-linear dynamic regret, i.e., the time-averaged regret approaches zero as T increases. Thus, as long as we choose μ such that $1 < \mu \le 2$, AdaSparse achieves both sub-linear dynamic regret in terms of delay and the optimal rate in terms of convergence.*

7 Numerical Experiments

We evaluate the efficacy of AdaSparse and LC-AdaSparse for classification task on MNIST [13], CIFAR-10 [12], and ImageNette (a 10-class subset of ImageNet [6]).

For comparison, we consider the following benchmarks:

- **Top-k [1]:** All devices send k entries with the largest magnitudes and use standard error feedback.
- **UniSparse [19]:** All devices use a common, fixed threshold for sparsification, with the standard error-feedback mechanism. This approach can be viewed as a special case of AdaSparse with a constant threshold, i.e., $\lambda_{m,t} = \lambda, \forall m, \forall t$. We refer to this approach as Uniform Sparsification.
- **VarReduced [23]:** This approach designs the sparsification by minimizing the expected number of entries sent by each device while limiting the sparsification variance.

Given that the problem formulation in (12) minimizes the overall delay while ensuring a certain level of convergence for the global loss function, we consider different levels of target accuracy as a measure for convergence, and compare the delay incurred by different methods when their test accuracy reaches the target level. Specifically, the delay is computed by $\sum_{t=0}^{T^\star-1}\sum_{m=1}^{M}\tau_{m,t}$, where $T^\star$ is the iteration at which the test accuracy reaches the target level.

For each target accuracy, we conduct hyperparameter tuning to select the optimal parameters for each approach that minimize the delay while achieving the target accuracy. Specifically, for Top-k, k is tuned, for UniSparse, λ is tuned, and for VarReduced, a parameter κ is tuned. For both AdaSparse and LC-AdaSparse, the parameter V is tuned. The best parameter values for each algorithm and target accuracy are provided in the GitHub repository of our implementation.[3]

7.1 Performance under Various Datasets

We consider $M = 10$ devices and assume the delay per coordinate is sampled from a uniform distribution, i.e., $d_{m,t} = d_{\max}c_{m,t}$, where $c_{m,t} \sim U[0,1]$ and $d_{\max} = 10^{-4}$ sec is the maximum delay for sending a coordinate. For LC-AdaSparse, we set $B = 25$ in all experiments.

7.1.1 MNIST Dataset. In MNIST, each sample is a grey-scaled handwritten digit image of size $\mathbb{R}^{28} \times \mathbb{R}^{28}$ pixels, with a label indicating its class. There are 60,000 training and 10,000 test samples. We consider training a multinomial logistic regression classifier with cross-entropy loss. The model for each class consists of 784 weights and a bias term, resulting in an overall model size of 7850.

An equal number of data samples from different classes are uniformly and randomly distributed among the devices. The batch size is set to 500. A training epoch is defined as one complete pass over the entire training dataset by all devices. In this setup, each epoch comprises 12 communication iterations, as the local data sets are 12 times larger than the batch size. We set the number of training epochs to 50 and thus the number of iterations is $T = 600$. The learning rate is constant during training and set to $\gamma_t = \gamma = 0.5$. The SGD optimizer with a weight decay of 10^{-4} is used for training.

We consider five target accuracy percentage values: 70, 75, 80, 85, and 90. For both AdaSparse and LC-AdaSparse, ϵ is set to 2.

[3] https://github.com/faezemoradik/AdaSparse.git

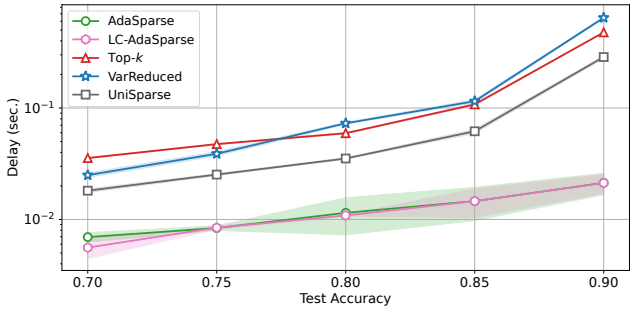


Figure 1: Delay vs. test accuracy for MNIST (i.i.d. data).

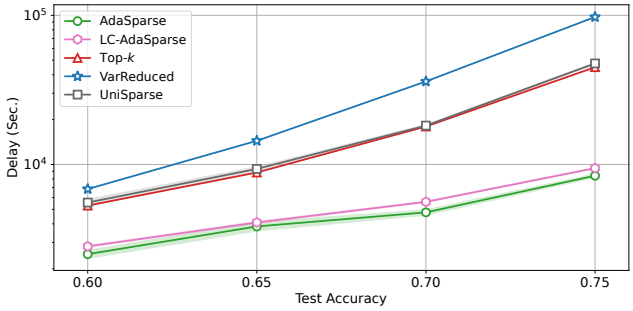


Figure 2: Delay vs. test accuracy for CIFAR-10 (non-i.i.d. data).

Figure 1 illustrates the delay incurred by each approach over the test accuracy. The results are averaged over three realizations, with the shaded area around each curve representing the 95% confidence intervals. As shown, the delay achieved by LC-AdaSparse is close to that of AdaSparse, and both methods achieve the same accuracy with significantly less delay than the benchmarks. Notably, at 90% accuracy, both LC-AdaSparse and AdaSparse reduce the delay by more than 10-fold compared with the minimum delay achieved among the benchmarks. Additionally, across other target accuracies, they result in a delay reduction of more than 2.5 times compared with the best-performing benchmark.

7.1.2 CIFAR-10 Dataset. In CIFAR-10, each data sample consists of a colored image of size $\mathbb{R}^3 \times \mathbb{R}^{32} \times \mathbb{R}^{32}$ and a label indicating the class of the image. There are $50,000$ training and $10,000$ test samples. We train a Residual Network (ResNet-9 [11] with about 2.5 million parameters) model with the Mish activation function [16] and group normalization instead of batch normalization [11]. We use the cross-entropy loss function.

To evaluate the effectiveness of the proposed methods under the non-i.i.d. data distribution, we distribute CIFAR-10 across devices such that each device contains data from only two classes and holds 5000 samples. We set the batch size to 50 and train for 60 epochs, resulting in $T = 6000$. We use the Nesterov-Adam optimizer with $(\beta_1, \beta_2) = (0.9, 0.999)$ and learning rate $\gamma = 10^{-4}$.

We consider four target accuracy levels: 60%, 65%, 70%, and 75%, and set $\epsilon = 15$ for both AdaSparse and LC-AdaSparse. Figure 2 presents the delay incurred by each approach versus the test accuracy. Both AdaSparse and LC-AdaSparse reduce the delay by more than 1.8-fold, 2-fold, 3-fold, and 4-fold for accuracies of 60%, 65%, 70%, and 75%, respectively. These results indicate that the proposed methods significantly reduce the communication latency.

7.1.3 ImageNette Dataset. In ImageNette, each sample consists of a colored image of size $\mathbb{R}^3 \times \mathbb{R}^{160} \times \mathbb{R}^{160}$. It contains $9,468$ training samples and $3,925$ test samples. We use the same ResNet-9 architecture as in Section 7.1.2 with the cross-entropy loss function.

The training data samples are distributed evenly across devices. We set the batch size to 50 and perform training over 60 epochs, resulting in $T = 1080$. We use the Nesterov-Adam optimizer with $(\beta_1, \beta_2) = (0.9, 0.999)$, and set the learning rate to $\gamma = 10^{-4}$.

We consider five levels of target percentage accuracy: 55, 60, 65, 70, and 75. For both AdaSparse and LC-AdaSparse, we set $\epsilon = 15$.

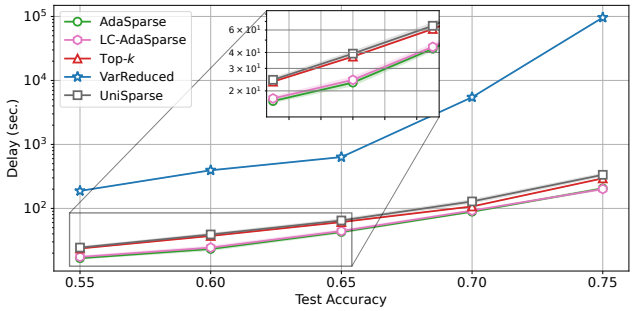


Figure 3: Delay vs. test accuracy for ImageNette (i.i.d. data).

Figure 3 illustrates the delay incurred by each approach against the test accuracy. Again, we observe that both AdaSparse and LC-AdaSparse achieve significantly reduced delay for the same accuracy levels. Specifically, they reduce the delay compared with the best-performing benchmark, Top-k, by 29%, 37%, 30%, 17%, and 30% at accuracies of 55%, 60%, 65%, 70%, and 75%, respectively.

7.2 Impact of System Heterogeneity

To further investigate the effect of system heterogeneity on the performance of the proposed algorithms, we use the same setting as in Section 7.1, except that the delay per coordinate, $d_{m,t} = d_{\max} c_{m,t}$, is sampled from a Beta distribution with parameters $\beta = \alpha$, i.e., $c_{m,t} \sim \text{Beta}(\alpha, \alpha)$, where $\alpha \geq 0$. Under this setting, the mean of $d_{m,t}$ is $\frac{d_{\max}}{2}$, and the variance is $\frac{d_{\max}^2}{4(2\alpha+1)}$. Therefore, decreasing the value of α increases the variance, which in turn amplifies the heterogeneity of the system. In the special case, where $\alpha = 1$, $\text{Beta}(\alpha, \alpha)$ reduces to $\text{U}[0, 1]$ which is the same as the setting in the previous subsections. We evaluate the performance of different approaches under various values of α.

We train ResNet-20 [9] with approximately $200,000$ parameters using the cross-entropy loss. The CIFAR-10 data is distributed across devices in an i.i.d. manner, with each device containing 5000 samples. The batch size is set to 50, and the training is conducted over 45 epochs, resulting in $T = 4500$. The learning rate is set to $\gamma = 0.01$, and the SGD optimizer with a momentum of 0.9 and a weight decay of 10^{-4} is used for training. We consider a target accuracy of 70% and set $\epsilon = 2$ for both AdaSparse and LC-AdaSparse.

Table 1: Delay vs. α values for CIFAR-10 with i.i.d. data.

	Average delay in sec. for reaching accuracy of 70%		
Methods	$\alpha = 0.2$	$\alpha = 0.5$	$\alpha = 0.8$
Top-k	47.38 ± 0.14	47.46 ± 0.08	47.62 ± 0.09
UniSparse	81.88 ± 1.83	81.93 ± 1.66	82.18 ± 1.75
VarReduced	2446.80 ± 8.27	2449.17 ± 3.38	2457.02 ± 4.29
AdaSparse	1.86 ± 0.06	20.28 ± 0.45	31.09 ± 1.52
LC-AdaSparse	1.95 ± 0.17	21.21 ± 0.61	31.75 ± 0.93

Table 1 shows the delay incurred by each approach versus the α value. The results are averaged over three realizations, with 95% confidence intervals. As shown, as the heterogeneity among devices increases (lower values of α), the delay incurred by both AdaSparse and LC-AdaSparse decreases. This is because they place less emphasis on slow devices with high per-coordinate delays by setting a high sparsification threshold for them, reducing the number of entries transmitted from their signal. Conversely, for faster devices with low $d_{m,t}$, it assigns a lower sparsification threshold, allowing them to transmit more entries. Specifically, when $\alpha = 0.2$, AdaSparse and LC-AdaSparse reduce the delay by more than 25 times, while for $\alpha = 0.5$ and $\alpha = 0.8$, the delay is reduced by more than 2 times and 1.5 times, respectively, compared with the best-performing benchmark. In contrast, we observe that the delay for other benchmark approaches remains approximately constant as α varies. This is because these benchmarks do not account for the heterogeneous per-coordinate delay values of devices during training. Instead, their performance depends only on the mean value of $d_{m,t}$, which is constant and equal to $\frac{d_{\max}}{2}$ across all three settings.

8 Conclusion

In this work, we have investigated the adaptive design of sparsification in a distributed learning system with dynamic communication conditions among devices. Unlike previous works, we explicitly aim to minimize the overall communication delay while ensuring a specified level of convergence for the loss function. The optimization problem depends on unknown future information, which necessitates an online solution. We propose AdaSparse, a distributed online algorithm, and its low-complexity approximation, LC-AdaSparse, which features linear computational complexity. Through novel drift bounding techniques, we show AdaSparse can achieve both sub-linear dynamic regret in delay and optimal rate of convergence. Numerical experiments on image classification tasks using various models and datasets have shown that our approach effectively reduces communication delay compared with the benchmarks, without compromising the learning performance.

Acknowledgments

This work was supported in part by Ericsson, the Natural Sciences and Engineering Research Council of Canada, and Mitacs. We thank the anonymous reviewers and the shepherd for valuable feedback.

References

[1] Alham Fikri Aji and Kenneth Heafield. 2017. Sparse communication for distributed gradient descent. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP).*

[2] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS).*

[3] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cédric Renggli. 2018. The convergence of sparsified gradient methods. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS).*

[4] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. 2019. Qsparse-local-SGD: Distributed SGD with quantization, sparsification and local computations. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS).*

[5] Ali Bereyhi, Ben Liang, Garya Boudreau, and Ali Afana. 2024. Novel Gradient Sparsification Algorithm via Bayesian Inference. In *IEEE 34th Int. Workshop on Machine Learning for Signal Processing.*

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *IEEE Conf. on Computer Vision and Pattern Recognition.*

[7] Jinyang Guo, Jianyu Wu, Zining Wang, Jiaheng Liu, Ge Yang, Yifu Ding, Ruihao Gong, Haotong Qin, and Xianglong Liu. 2024. Compressing Large Language Models by Joint Sparsification and Quantization. In *Proc. Int. Conf. on Machine Learning (ICML).*

[8] Pengchao Han, Shiqiang Wang, and Kin K Leung. 2020. Adaptive gradient sparsification for efficient federated learning: An online learning approach. In *Proc. IEEE Int. Conf. on Distributed Computing Systems (ICDCS).*

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition.*

[10] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Ion Stoica, Raman Arora, et al. 2019. Communication-efficient distributed SGD with sketching. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS).*

[11] Helena Klause, Alexander Ziller, Daniel Rueckert, Kerstin Hammernik, and Georgios Kaissis. 2022. Differentially private training of residual networks with scale normalisation. *arXiv preprint arXiv:2203.00324* (2022).

[12] Alex Krizhevsky and Geoffrey E. Hinton. 2009. *Learning multiple layers of features from tiny images.* Master's thesis. University of Toronto.

[13] Yann LeCun, Corinna Cortes, and Christopher J. Burges. 1998. The MNIST Database of Handwritten Digits. *[Online]. Available: http://yann.lecun.com/exdb/mnist/* (1998).

[14] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. 2018. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. In *Int. Conf. on Learning Representations (ICLR).*

[15] Ouiame Marnissi, Hajar El Hammouti, and El Houcine Bergou. 2024. Adaptive sparsification and quantization for enhanced energy efficiency in federated learning. *IEEE Open J. Commun. Soc.* (2024).

[16] Diganta Misra. 2019. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681* (2019).

[17] Michael J. Neely. 2010. *Stochastic Network Optimization with Application to Communication and Queueing Systems.* Morgan & Claypool, CA, USA.

[18] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. 2020. FetchSGD: Communication-efficient federated learning with sketching. In *Proc. Int. Conf. on Machine Learning (ICML).*

[19] Atal Sahu, Aritra Dutta, Ahmed M Abdelmoniem, Trambak Banerjee, Marco Canini, and Panos Kalnis. 2021. Rethinking gradient sparsification as total error minimization. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS).*

[20] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. 2018. Sparsified SGD with memory. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS).*

[21] Shubham Vaishnav, Sarit Khirirat, and Sindri Magnússon. 2024. Communication-Adaptive Gradient Sparsification for Federated Learning with Error Compensation. *IEEE Internet Things J.* (2024).

[22] Bin Wang, Jun Fang, Hongbin Li, and Bing Zeng. 2023. Communication-efficient federated learning: A variance-reduced stochastic approach with adaptive sparsification. *IEEE Trans. Signal Process.* 71 (2023), 3562–3576.

[23] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. 2018. Gradient sparsification for communication-efficient distributed optimization. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS).*

[24] Weiying Xie, Haowei Li, Jitao Ma, Yunsong Li, Jie Lei, Donglai Liu, and Leyuan Fang. 2024. JointSQ: Joint Sparsification-Quantization for Distributed Learning. In *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR).*

[25] Mingchao Yu, Zhifeng Lin, Krishna Narra, Songze Li, Youjie Li, Nam Sung Kim, Alexander Schwing, Murali Annavaram, and Salman Avestimehr. 2018. GradiVeQ: Vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS).*

[26] Zhaorui Zhang and Choli Wang. 2022. MIPD: An adaptive gradient sparsification framework for distributed DNNs training. *IEEE Trans. on Parallel and Distributed Syst.* 33, 11 (2022), 3053–3066.

[27] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. 2010. Parallelized stochastic gradient descent. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS).*