Kimad: Adaptive Gradient Compression
with Bandwidth Awareness

Jihao Xin*
KAUST

Ivan Ilin*
KAUST

Shunkang Zhang
HKUST

Marco Canini
KAUST

Peter Richtárik
KAUST

ABSTRACT

In distributed training, communication often emerges as a bottleneck. In response, we introduce Kimad, a solution that offers adaptive gradient compression. By consistently monitoring bandwidth, Kimad refines compression ratios to match specific neural network layer requirements. Our exhaustive tests and proofs confirm Kimad's outstanding performance, establishing it as a benchmark in adaptive compression for distributed deep learning.

CCS CONCEPTS

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

KEYWORDS

Distributed Training, Gradient Compression

1 INTRODUCTION

Deep learning has steadily emerged as a transformative paradigm, demonstrating profound results in various domains. With its growth, there's been an explosion in the size of models and datasets. This upsurge in complexity often demands expansive computational resources, prompting researchers to adopt distributed training.

The Graphics Processing Unit (GPU) has emerged as a cornerstone in the realm of deep learning model training, fundamentally altering the landscape of artificial intelligence research and applications. The latest advancements in GPU technology, exemplified by the state-of-the-art models such as the Ampere and Hopper [9, 10], exhibit unprecedented computational power and speed up the training by up to 16×. However, it is noteworthy that the acquisition of these cutting-edge GPUs comes at a considerable financial cost, more than $200,000 for a single DGX A100.
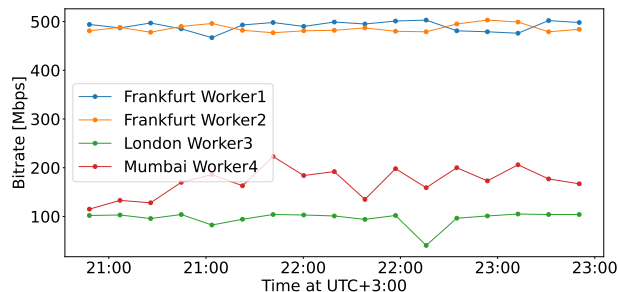
*Authors contributed equally to this work.

Figure 1: EC2 bandwidth with TCP server at Frankfurt.

In this scenario, researchers increasingly turn to cloud-based computational resources for model training due to their flexible pricing models, variety of hardware, and ease of scaling computational resources. However, the bandwidth variability problem in cloud-based deep learning training poses a substantial challenge to the efficient execution of large-scale machine learning tasks [1, 20, 27]. The bandwidth fluctuations, influenced by factors such as network congestion and competing workloads, lead to inconsistent performance during training. Figure 1 shows an example of bandwidth discrepancy measured at AWS EC2 with a TCP server in Frankfurt receiving simultaneously from 4 workers using IPerf3. While the existing framework CGX [21] has made strides by offering a comprehensive approach that integrates widely adopted gradient compression techniques and strikes a balance between accuracy and compression ratio, it failed to address dynamic bandwidth considerations. DC2 [1] achieves adaptive compression by inserting a shim layer between the ML framework and network stack to do real-time bandwidth monitoring and adjust the compression ratio. However, this approach is model-agnostic which cannot be used together with other application-level optimization. In addition to bandwidth adaptivity, numerous researchers are investigating how to capitalize on the diverse structural nature present across network layers in order to enhance compression ratios [3, 8]. However, these studies solely address the static nature of network structures and assume an ideally stable network connection, a scenario seldom encountered in real-world deployments.

In light of these findings, we introduce Kimad: an adaptive gradient compression system designed to be aware of both bandwidth changes and model structures. The comprehensive designs are depicted in Figure 2. Kimad deploys a runtime bandwidth monitor and a compression module on each worker and server. Throughout the training phase, the bandwidth monitor gauges communication delays using historical statistics. Subsequently, the compression

module utilizes the estimated bandwidth to compute the compression budget for the entire model. It then refines the layer-wise compression ratios while adhering to the overarching compression budget constraints.

In essence, we advance the following contributions:

- We propose Kimad, a general framework for gradient compression adaptive to bandwidth changes.
- We further propose Kimad+, which incorporates an adaptive compression ratio tailored to layers to minimize compression errors.
- We expand the error feedback framework to elucidate its functionality within the context of Kimad.
- We evaluate from a convex function to a deep model, revealing that Kimad can accelerate the training while maintaining convergence.

2 BACKGROUND AND RELATED WORK

2.1 Data Parallelism

Data parallelism is a widely used strategy to solve the distributed training problem, which can be formulated as (1).

$$\min_{x \in \mathbb{R}^d} \left\{ f(x) := \sum_{m=1}^{M} w_m f_m(x) \right\} \tag{1}$$

$x \in \mathbb{R}^d$ corresponds to the parameters of a model, $[M] := \{1, \ldots, M\}$ is the set of workers (e.g. GPUs, IoT devices) and $w_1, \ldots, w_M$ are non-negative weights adding up to 1 (for example, the weights can be uniform, i.e., $w_m = \frac{1}{M}$ for all m). Further, $f_m(x) := \mathrm{E}_{\xi \sim \mathcal{D}_m} [\ell(x, \xi)]$ is the empirical loss of model x over the training data $\mathcal{D}_m$ stored on worker m, where $\ell(x, \xi)$ is the loss of model x on a single data point ξ.

In data parallelism, each worker keeps a copy of the model and a partition of the dataset. The gradients computed on each worker are then communicated to aggregate and update the model. We provide the general formulation to solve (1) in Appendix A.

In this work, we predominantly focus on the Parameter-Server model. Our choice is driven by its inherent capability to efficiently handle sparse updates [12, 16], and its widespread adoption in environments with shared bandwidth like federated learning. While our emphasis is on the PS architecture with Data Parallelism, we posit that the adaptivity innovations we introduce can seamlessly integrate and offer value to the Peer-to-Peer architecture and model parallelism as well.

2.2 Gradient Compression

Relying on the nature that deep learning training can converge despite lossy information, gradient compression is a popular approach to speed up data-parallel training [36]. During back-propagation, gradients will be compressed before communication with the server, and the server will decompress the gradients prior to aggregating them; thus the communication cost can be largely reduced. Additionally, the server can distribute the model using compression as well. Gradient compression techniques can be generally categorized into three classes:

- **Sparsification** [5, 17, 28, 29, 31]: Selectively retaining elements in gradients while zeroing others. This includes methods like TopK (selecting the K largest absolute value elements) and RandK (randomly selecting K elements).
- **Quantization** [4, 13, 22, 26, 35]: Reducing data precision to fewer discrete values. Deep learning frameworks often use Floating Point 32 (FP32) for gradients, which can be compressed to formats like FP16, UINT8, or even 1 bit [26].
- **Low-Rank Decomposition** [30]: Approximating gradients by breaking them down into lower-rank matrices, reducing their size as $A \approx U \cdot V^T$, where A is the original matrix, and U and V are lower-rank matrices.

Adaptive compression. Adaptive compression is an emerging area to study how to apply gradient compression efficiently with different compression levels [1, 2]. Gradient compression is traditionally used in an intuitive way: Given a compressor $C : \mathbb{R}^d \to \mathbb{R}^d$, gradients are compressed with a static strategy where the same compression ratio is used for each layer and across the whole training procedure. However, gradient compression has a different impact on different training stages. For instance, Accordion [2] selects between high and low compression levels by identifying the critical learning regimes. Furthermore, it is incumbent upon gradient compression methodologies to account for the diverse attributes of individual layers. For example, Egeria [34] methodically freezes layers that have achieved convergence during training. L-Greco [3] uses dynamic programming to adjust layer-specific compression ratios given the error budget, reducing overall compressed size. Moreover, researchers should also take the system architecture into consideration. Notably, FlexReduce [18] proposes that the communication protocol can be split into different portions unevenly based on the communication hierarchy.

2.3 Error Feedback

Error feedback (EF), also referred to as error compensation, is a widely adopted method for ensuring convergence stability in distributed training of supervised machine learning models. It is particularly effective when combined with biased compressors such as TopK. EF was originally introduced as a heuristic [26]; then theoretical guarantees were proposed [5, 28]. More recently, EF21 [11, 23, 24] provides theoretical analysis for distributed settings and achieves a state-of-the-art $O(1/T)$ convergence rate. We integrate EF21 into Kimad to achieve better convergence.

2.4 Bandwidth Monitoring

Bandwidth monitoring is critical in network management, especially in cloud-based scenarios. It addresses the need to monitor data transfer rates between computational nodes during training, ensuring optimal communication efficiency. Existing works [1, 6, 7] allow us to estimate the bandwidth changes by utilizing a collection of the network-level communication properties such as the latency. Particularly, adaptive strategies [33, 37] such as dynamic synchronization algorithms or buffering mechanisms can alleviate the effects of bandwidth fluctuation.

3 METHODOLOGY

We propose Kimad, an adaptive compression framework to accommodate varying bandwidth and model structures. Kimad continuously monitors the bandwidth and dynamically adjusts the volume

of communication size in each round for every machine. For instance, if the bandwidth B_m^k for machine m at step k becomes limited compared to other devices, we instruct machine m to employ a suitable compressor to reduce the size of the update vector[1] u_m^k with the goal of ensuring that this machine does not become a straggler. Additionally, we present Kimad+, an extension of Kimad, which fine-tunes the compression ratio differently across layers. Kimad+ involves an additional step that introduces some computational overhead and is recommended for use when there is surplus computational capacity available (i.e., when communication is the most severe bottleneck).

As Figure 2 shows, to train a deep learning task, the end users need to inform Kimad of t, which is the **time budget** for a single communication round (a step). The server and each worker will determine the compression strategy locally without knowing global information. Kimad requires a bandwidth monitor, which is deployed on each worker and server, and will continuously monitor the network behavior and estimate the current bandwidth. Kimad will calculate how many bits need to be communicated at each step based on the bandwidth, which we call the **compression budget** denoted as c. The blue arrows in Figure 2 further represent Kimad+, which allocates the compression budget to each layer to minimize the compression error and thus improve accuracy.

Algorithm 1 formulates the general version of Kimad. The algorithm starts with the server broadcasting the latest compressed update $C^k(x^k - \hat{x}^{k-1})$, then each worker will calculate the update by A_m^{update} and upload the compressed update $C_m^k(u_m^k - \hat{u}_m^{k-1})$. Afterward, the server will update the model x^k by the aggregated update vector. The **core of the algorithm** is A^{compress}, which selects a compressor from Ω in an **adaptive** manner, based on the model information and current bandwidth estimation B_m^k. To recover accuracy, we apply bidirectional EF21, therefore, both server and workers maintain two estimators: $\hat{u}_m^k$ and $\hat{x}^k$, and only the server stores the global model x^k. We put a detailed version of the Kimad algorithm in Algorithm 3 in Appendix B.

3.1 Kimad: Bandwidth Adaptivity

With a user-specified time budget t, the target of Kimad is to limit the training time at each step within t time units while communicating as much information as possible.

In our work, we examine asymmetric networks, e.g., the up-link and down-link bandwidth can be different, and the bandwidth varies among workers. We apply bidirectional compression, i.e., both workers and server send compressed information.

We break down the time cost of worker m at step k as:

$$t = T_{comm}^u + T_{comp} + T_{comm}^d$$

We abstract the computation time of a step as T_{comp} which is assumed to be constant across a training task. For the uplink communication, we define $T_{comm}^u = \frac{c}{B_m^k}$. For the downlink, we define $T_{comm}^d = \alpha \frac{c}{B_m^k}$ and α is the coefficient of broadcasting congestion which can be simply set to 1 assuming no congestion. Therefore, for

---

[1]The update vector is the gradient under basic SGD setting. When applying EF21, it is the difference between the gradient estimation and the real gradient.

Algorithm 1 Kimad: Adaptive Gradient Compression with Bandwidth Awareness

1: **Input:** loss ℓ; weights w_m; datasets $\mathcal{D}_m$; model updating algorithm A_m^{update} on each worker $m \in [M]$; set of compressors Ω; compressor-selection algorithm A^{compress} on server and workers; model $x^0 \in \mathbb{R}^d$ known by the server; initial model estimator $\hat{x}^{-1} \in \mathbb{R}^d$ and initial update estimators $\hat{u}_m^{-1} \in \mathbb{R}^d$, known by the workers and the server; single round time budget $t > 0$; learning rate schedule $\{\gamma^k\} > 0$ for iterations $k \geq 0$.
2: **for each communication round** $k = 0, 1, 2, \ldots$ **do**
   //Server:
3:    Estimate B^k at communication round k
4:    Select compressor: $C^k = A^{\text{compress}}(\Omega, x^k, \hat{x}^{k-1}, B^k, t)$
5:    Update model estimator: $\hat{x}^k = \hat{x}^{k-1} + C^k(x^k - \hat{x}^{k-1})$
6:    Broadcasts the compressed vector $C^k(x^k - \hat{x}^{k-1})$ to all workers $m \in [M]$
   //Workers:
7:    **for each worker** $m = 1, 2, \ldots, M$ **in parallel do**
8:       Update model estimator:
$$\hat{x}^k = \hat{x}^{k-1} + C^k(x^k - \hat{x}^{k-1})$$
9:       Calculate update: $u_m^k = A_m^{\text{update}}\left(\hat{x}^k, \ell, \mathcal{D}_m\right) \in \mathbb{R}^d$
10:      Select compressor:
$$C_m^k = A^{\text{compress}}(\Omega, u_m^k, \hat{u}_m^{k-1}, B_m^k, t)$$
11:      Upload the compressed vector $C_m^k(u_m^k - \hat{u}_m^{k-1})$ to the server
12:   **end for**
13:   //Server:
14:   Aggregate all update estimators:
$$\hat{u}_m^k = \hat{u}_m^{k-1} + C_m^k(u_m^k - \hat{u}_m^{k-1}), \qquad m \in [M]$$
15:   Updates the model via $x^{k+1} = x^k - \gamma^k \sum_{m=1}^M w_m \hat{u}_m^k$
16: **end for**

---

simplicity, and without loss of generality, we only consider varying T_{comm} to simulate various scenarios.

When communication is triggered, Kimad will read the current bandwidth from the bandwidth estimator and use it to calculate (with negligible computation overhead) the compression budget as:

$$c = B_m^k \frac{t - T_{comp}}{2}. \tag{2}$$

3.2 Kimad+: Layer Adaptivity

With a predefined compression budget, Kimad+ can dynamically allocate the compression ratios of individual layers in a non-uniform manner. This optimization aims to enhance performance while ensuring that the cumulative compression ratio remains within the allocated budget. We start by formulating it as an optimization problem as:

$$\min \varepsilon^k = \sum_{i=1}^l \varepsilon^i, \quad \text{subject to} \sum_{i=1}^\ell b_{ij_i^k} \leq c \tag{3}$$

The target is to minimize the total error ε^k across layers caused by compression, with compressed size constrained by the compression
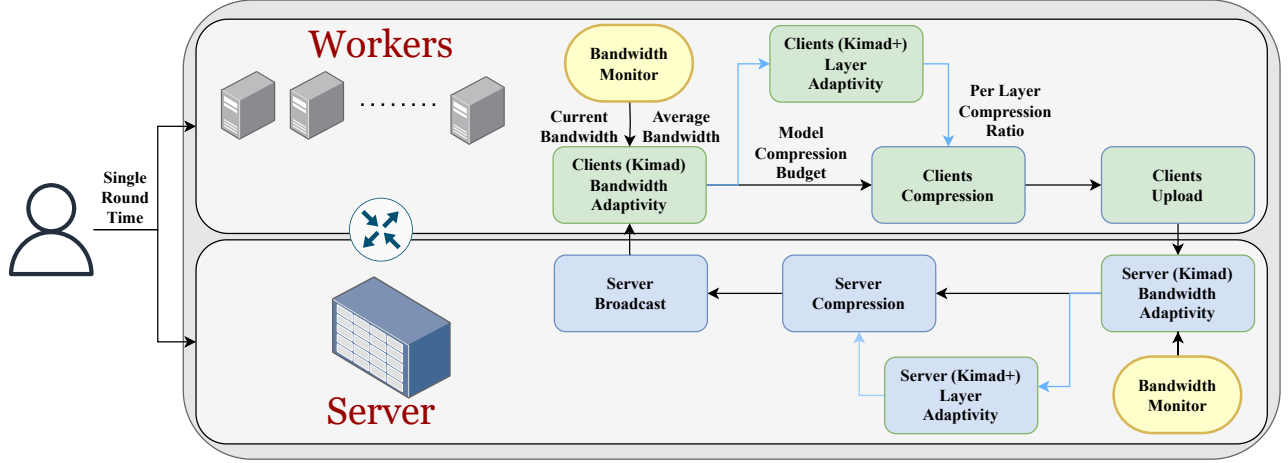
Figure 2: Kimad framework.

budget. We consider the standard Euclidean norm (l_2-norm) as the error indicator defined by:

$$\varepsilon^k = \left\| \hat{u}^k - u^k \right\|^2 = \sum_{i=1}^{\ell} \left\| \hat{u}_i^k - u_i \right\|^2 \qquad (4)$$

However, the relation between the compression error and compressed size is not deterministic, and the search space of the compression ratio is continuous. As a result, finding an analytical solution for this optimization problem is not feasible. To tackle this challenge, we employ a discretization approach, narrowing down the compression ratio search space. Specifically, for each layer, Kimad+ restricts its choice of compression ratio to a discrete set $\{1, 2, \ldots, w\}$. Therefore, (3) can be written as:

$$\min_{j_1^k, \cdots, j_\ell^k} \qquad \varepsilon^k(j_1^k, \ldots, j_\ell^k) = \sum_{i=1}^{l} \varepsilon^i(j_i^k)$$

$$\text{subject to} \qquad j_1^k \in \{1, \ldots, w_1\}, \cdots, j_\ell^k \in \{1, \ldots, w_\ell\}$$

$$\sum_{i=1}^{\ell} b_{ij_i^k} \le c,$$

We adopt the idea from L-Greco [3] to formulate it as a knapsack problem. In contrast to L-Greco, Kimad+ uses the compression budget c as the knapsack size and the compression error as the weight. Then, Kimad+ uses dynamic programming to solve the knapsack problem. The time complexity is $O(NKD)$ where N is the number of layers, K represents the possible compression ratios, and D is the discretization factor for the error. We give the algorithm details in Appendix C.

3.3 Error Feedback

We apply error feedback within Kimad. To the best of our knowledge, EF21 [23] is one of the most effective EF methods. We adapt EF21 and extend it in a layer-wise fashion. However, while EF21 is analyzed using a constant step size, our theory here allows the step size to depend on the layer i and on the iteration k. Below we give the theoretical result; the proof is in Appendices D, E, and F.

We initialize by choosing $x_i^0, \hat{u}_i^0 \in \mathbb{R}^{d_i}$ for $i = 1, \ldots, \ell$, $x^0 = (x_1^0, \ldots, x_\ell^0) \in \mathbb{R}^d$. After this, for $k = 0, 1, \ldots$ we iterate:

$$x_i^{k+1} = x_i^k - \gamma_i^k \hat{u}_i^k, \quad i = 1, \ldots, \ell, \qquad (5)$$

$$\hat{u}_i^{k+1} = \hat{u}_i^k + C_i^k(\nabla_i f(x^{k+1}) - \hat{u}_i^k), \quad i = 1, \ldots, \ell, \qquad (6)$$

$$x^{k+1} = (x_1^{k+1}, \ldots, x_\ell^{k+1}), \qquad (7)$$

where $\gamma_i^k \ge 0$ are step sizes and $C_i^k : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$ are compressors for all $i = 1, \ldots, \ell$ and $k \ge 0$. Let $\hat{u}^k = (\hat{u}_1^k, \ldots, \hat{u}_\ell^k)$.

Theory. We state our main theorem below. We assume the model is layer-smooth (Appendix D.1) and the result extends to global-smooth (Appendix D.2).

Theorem 1. *Consider Algorithm (5)–(7). Assume f is lower bounded by $f^{\inf} \in \mathbb{R}$, and let the layer smoothness (Assumption 1) and global smoothness (Assumption 2) conditions hold. Assume that $C_i^k \in \mathbb{C}^{d_i}(\alpha_i)$ for all $k \ge 0$ and all $i \in [\ell]$, where $\alpha_i \in (0, 1]$. Choose any $\zeta_1, \ldots, \zeta_\ell > 0$ such that $(1 - \alpha_i)(1 + \zeta_i) < 1$ for all i, and define*

$$\theta_i := 1 - (1 - \alpha_i)(1 + \zeta_i), \quad \beta_i := (1 - \alpha_i)\left(1 + \zeta_i^{-1}\right) \qquad (8)$$

and $\theta := \min_i \theta_i$. Choose any $w_1, \ldots, w_\ell > 0$ and let the step sizes be chosen via $\gamma_i^k \equiv \gamma w_i$ for all $k \ge 0$, where $\gamma > 0$ satisfies

$$\gamma^2 \frac{w_i\left(\max_i \frac{w_i}{\delta_i}\right)\left(\max_i \delta_i \beta_i\right) L^2}{\theta} + \gamma L_i w_i \le 1. \qquad (9)$$

Then

$$\frac{1}{K} \sum_{k=0}^{K-1} \left(\sum_{i=1}^{\ell} w_i \mathrm{E}\left[\left\| \nabla_i f(x^k) \right\|^2 \right] \right) \le$$

$$\frac{2(f(x^0) - f^{\inf})}{\gamma K} + \frac{\left(\max_i \frac{w_i}{\delta_i}\right) \sum_{i=1}^{\ell} \delta_i \left\| \hat{u}_i^0 - \nabla_i f(x^0) \right\|^2}{\theta K}.$$

4 EVALUATION

We begin our evaluation by initially performing synthetic experiments to showcase the efficiency of our proposed Kimad method, particularly to demonstrate that EF21 can work with compression
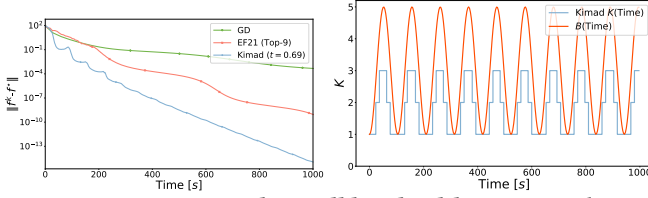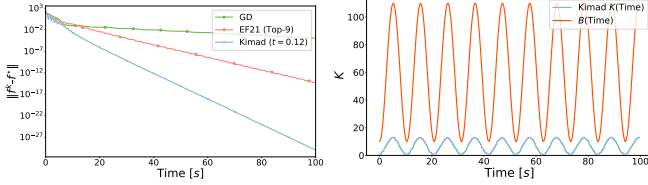
Figure 3: Extremely small bandwidth: $B_{max} << d$.

Figure 4: Small bandwidth: $B_{max} < d$.

Figure 5: Oscillation between small and high bandwidth.

Figure 6: High bandwidth with small oscillations (almost no adaptation).

ratio adaptive to bandwidth. The synthetic experiments are done with a simple quadratic function f which is lower bounded by 0, and has layer smoothness (Appendix D.1) and global smoothness (Appendix D.2). This function fits the theory assumptions and allows us to fine-tune the learning rates for all compression ratios and time budget t at an affordable cost. Subsequently, we present results from more practical tasks, demonstrating that Kimad is applicable to distributed deep learning training. We also conduct an evaluation of Kimad+ to substantiate its superior capabilities of reducing compression error compared to Kimad, all while maintaining the same communication cost. The evaluation is simulation-based, running as a Parameter Server architecture with dynamic asymmetric bandwidth. We use TopK with fixed K as the default compression method. The simulator is tested with Python 3.9.15, and Pytorch 1.13.1.

4.1 Synthetic Experiments

For now, we consider only one direction; e.g., the down-link (server to worker) communication cost can be neglected. So, there is only an up-link bandwidth cost. We simulate the bandwidth oscillation with a sinusoid-like function as Figure 3.

We start our experiments in a single-worker setup to optimize a quadratic function. So, $M = 1$, $w_1 = 1$, $\mathcal{D}_1 = a = (a_1, \ldots, a_d)$, $a_i > 0$ for all i and $f_1(x) := E_{\xi \sim \mathcal{D}_m} [\ell(x, \xi)] = \ell(x, \xi) = \frac{1}{2} \sum_{i=1}^{d} a_i x_i^2$, where $d = 30$. Hence $f(x)$ in problem (1) can be written in the following form:

$$f(x) = \frac{1}{2} \sum_{i=1}^{d} a_i x_i^2 \tag{10}$$

Previous works [11, 23, 24] show that EF21 can improve convergence rates in federated learning setups, particularly for biased compressors such as TopK. We now demonstrate that EF21 can also be used to improve performance seamlessly with Kimad. For a fair comparison, it's crucial to fine-tune all hyperparameters for each method. For EF21 with TopK, we systematically explored various K values and selected the one that performed the best for comparison with Kimad. However, Kimad doesn't require us to determine the best K since it adapts to the available bandwidth dynamically. Instead, we focus on optimizing the time budget parameter t and
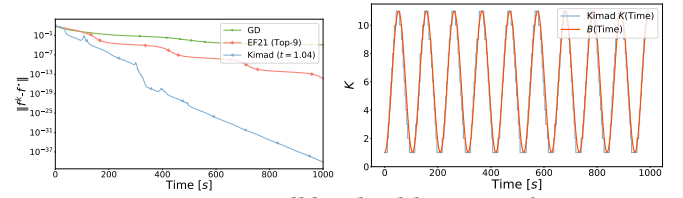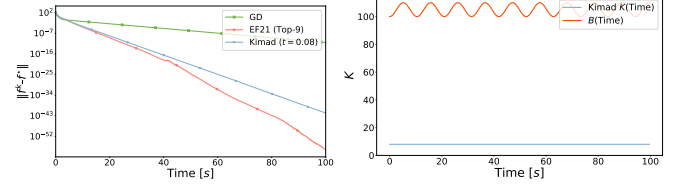
fine-tuning Kimad in conjunction with EF21. We compare performance among Kimad, EF21, and set the standard gradient descent (GD) as the baseline.

As Figure 3 shows, Kimad can be much faster than the best EF21. We achieved these results because Kimad adapted the compress ratio depending on the bandwidth to be as effective as possible. These results are consistent over different bandwidth patterns: with small bandwidth ($B_{max} < d$) and high relative oscillation we can see great results because we gain more with using adaptive strategy (Figure 3 and Figure 4). As the amplitude of the bandwidth oscillations becomes higher, we still have improvements in performance (Figure 5). However, when the bandwidth is very high and the amplitude of its oscillations is low, we do not gain from adapting of compress ratio: there is no need to adapt because the bandwidth is not a bottleneck anymore (Figure 6).

4.2 Kimad on Deep Model

Setting. We train ResNet18 on Cifar10 for 100 epochs. We set $\{w_m = 1, \forall m \in M\}$, $\gamma = 0.01$, $\Omega = \{TopK | K > 0\}$, A^{update} calculates gradients with batch size = 128, random seed=21. We conduct 5 epochs warmup training, thus $\hat{u}_m^k$ and $\hat{x}^k$ are initialized as u_m^5 and x^5. Compression occurs on a per-layer basis, in accordance with common practice. We set $\alpha = 1$ for the downlink so that the compression budget can be calculated by $c = T_{comm} B_m^k$. We set $T_{comp} = \frac{ModelSize}{AverageBandwidth}$.[2] Our baseline is EF21 with fixed-ratio compression, which has the same overall communication size as Kimad but applies the same compression ratio across layers and steps.

Bandwidth. In our simulation, we model the dynamic bandwidth within the range of 30 Mbps to 330 Mbps using the function: $Bandwidth(time) = \eta \sin(\theta \cdot time)^2 + \delta$, where η, θ, δ are user-defined coefficients to adjust the changing frequency and amplitude. We assume the bandwidth between the server and each worker follows the same patterns with different noise. The dashed curve in Figure 7 shows the bandwidth pattern.

Communication adaptivity. Figure 7 depicts a single worker's communication size over time for different T_{comm}. The left y-axis
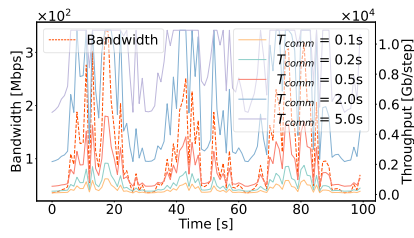
---

[2]Measured during the warmup epochs.
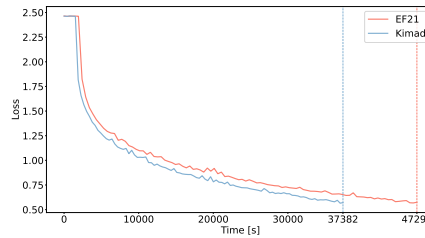
Figure 7: Communication throughput. $M = 4$ **workers.**

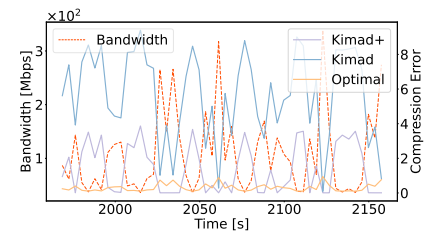**Figure 8: Loss curve.** $M = 4$ **workers,** $T_{comm} = 1.0s$.

Figure 9: Compression error. $M = 4$ **workers,** $T_{comm} = 1.0s$.

	1.0s	0.5s	0.2s	0.1s
EF21	486.1s	360.6s	284.2s	258.0s
Kimad	385.2s	285.2s	225.2s	205.2s

Table 1: Average step time across T_{comm}. $M = 4$ **workers.**

	2	4	8	16
EF21	79.70%	79.59%	79.23%	77.97%
Kimad	79.34%	79.75%	78.74%	77.97%

Table 2: Top5 accuracy across varying M. $T_{comm} = 1s$.

represents bandwidth, while the right represents communication size. The plateau at the top signifies the maximum uncompressed size. This graph illustrates Kimad's effective adaptation to changing bandwidth conditions, thus optimizing communication throughout.

Convergence. The loss curve in Figure 8 shows the comparison with EF21. Kimad finishes training faster while achieving the same final convergence.

Speedup. Table 1 lists the average time of one SGD step across different T_{comm}. In our setting, Kimad can generally save 20% training time for different communication budgets.

Scalability. Table 2 presents the Top5 accuracy on the evaluation set after 100 epochs. Kimad demonstrates comparable scalability to EF21 which maintains good accuracy levels with increasing number of workers.

4.3 Kimad+

Kimad+ minimizes the compression error while maintaining the same compression ratio as Kimad. We train Kimad and Kimad+ under the same setting as above with error discretization factor 1000 and compression ratio chosen from $\{x \mid x = 0.01 + k \cdot 0.02, \text{ where } k \in \mathbb{Z}, 0.01 \leq x \leq 1\}$. Figure 9 shows the compression error at one worker in a time frame, the optimal baseline is to select K with the whole model information. The compression error is negatively correlated with bandwidth, while Kimad+ can generally achieve lower compression error. We also observe that Kimad+ can achieve 1% higher accuracy than EF21 after the training.

5 LIMITATIONS AND FUTURE WORK

Kimad introduces a user-defined hyperparameter t, which is a trade-off between per-step time and accuracy and can also be adjusted dynamically. The learning rate can also be adjusted layer-wise

Besides, our work is not yet a fully implemented system. As the current experiments are simulation based, thus the implementation of monitor is trivial. We value the importance to integrate SOTA monitoring method to a complete work in the future. We can generalize the idea from splitting models to layers to blocks, where one block may contain many small layers. The computation overhead of Kimad+ is non-negligible and can be overlapped with communication. Moreover, LLM-targeted compression such as CocktailSGD [32] can also be considered.

6 CONCLUSION

We proposed Kimad, a bandwidth-aware gradient compression framework that comes with extended EF21. Kimad adapts the compression ratio based on the bandwidth and model characteristics; namely, each worker determines its local compression ratio considering its available bandwidth and time budget, and this ratio can be allocated to different layers in a non-uniform manner based on layer-wise sensitivity. We validated that Kimad can preserve the same convergence of fixed-ratio compression while saving communication time.

Acknowledgments

REFERENCES

[1] Ahmed M Abdelmoniem and Marco Canini. 2021. DC2: Delay-aware compression control for distributed machine learning. In *INFOCOM*.
[2] Saurabh Agarwal, Hongyi Wang, Kangwook Lee, Shivaram Venkataraman, and Dimitris Papailiopoulos. 2021. Accordion: Adaptive Gradient Communication via Critical Learning Regime Identification. In *MLSys*.
[3] Mohammadreza Alimohammadi, Ilia Markov, Elias Frantar, and Dan Alistarh. 2022. L-GreCo: An Efficient and General Framework for Layerwise-Adaptive Gradient Compression. arXiv:2210.17357 [cs.LG]
[4] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In *NeurIPS*.
[5] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Sarit Khirirat, Nikola Konstantinov, and Cédric Renggli. 2018. The Convergence of Sparsified Gradient Methods. In *NeurIPS*.
[6] Manu Anand. 2012. Cloud monitor: Monitoring applications in cloud. In *CCEM*.
[7] Eddy Caron, Luis Rodero-Merino, Frédéric Desprez, and Adrian Muresan. 2012. *Auto-scaling, load balancing and monitoring in commercial and open-source clouds*. Ph. D. Dissertation. INRIA.
[8] Changan Chen, Frederick Tung, Naveen Vedula, and Greg Mori. 2018. Constraint-aware deep neural network compression. In *ECCV*.

[9] Jack Choquette. 2023. NVIDIA Hopper H100 GPU: Scaling Performance. *IEEE Micro* 43, 3 (2023), 9–17.

[10] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro* 41, 2 (2021), 29–35.

[11] Ilyas Fatkhullin, Igor Sokolov, Eduard Gorbunov, Zhize Li, and Peter Richtárik. 2021. EF21 with Bells & Whistles: Practical Algorithmic Extensions of Modern Error Feedback. arXiv:2110.03294 [cs.LG]

[12] Jiawei Fei, Chen-Yu Ho, Atal Narayan Sahu, Marco Canini, and Amedeo Sapio. 2021. Efficient Sparse Collective Communication and its application to Accelerate Distributed Deep Learning. In *SIGCOMM*.

[13] Samuel Horváth, Chen-Yu Ho, Ludovit Horvath, Atal Narayan Sahu, Marco Canini, and Peter Richtárik. 2022. Natural Compression for Distributed Deep Learning. In *MSML*.

[14] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. 2019. First Analysis of Local GD on Heterogeneous Data. In *NeurIPS Workshop on Federated Learning for Data Privacy and Confidentiality*.

[15] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. 2020. Tighter Theory for Local SGD on Identical and Heterogeneous Data. In *AISTATS*.

[16] Soojeong Kim, Gyeong-In Yu, Hojin Park, Sungwoo Cho, Eunji Jeong, Hyeonmin Ha, Sanha Lee, Joo Seong Jeong, and Byung-Gon Chun. 2019. Parallax: Sparsity-aware Data Parallel Training of Deep Neural Networks. In *EuroSys*.

[17] Jakub Konečný and Peter Richtárik. 2018. Randomized distributed mean estimation: accuracy vs communication. *Frontiers in Applied Mathematics and Statistics* 4, 62 (2018), 1–11.

[18] Jinho Lee, Inseok Hwang, Soham Shah, and Minsik Cho. 2020. FlexReduce: Flexible All-reduce for Distributed Deep Learning on Asymmetric Network Topology. In *DAC*.

[19] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2020. On the Convergence of FedAvg on Non-IID Data. In *ICLR*.

[20] Liang Luo, Peter West, Arvind Krishnamurthy, Luis Ceze, and Jacob Nelson. 2020. PLink: Discovering And Exploiting Datacenter Network Locality For Efficient Cloud-based Distributed Training.

[21] Ilia Markov, Hamidreza Ramezanikebrya, and Dan Alistarh. 2022. CGX: Adaptive System Support for Communication-Efficient Deep Learning. In *Middleware*.

[22] Konstantin Mishchenko, Eduard Gorbunov, Martin Takáč, and Peter Richtárik. 2019. Distributed Learning with Compressed Gradient Differences. arXiv:1901.09269 [cs.LG]

[23] Peter Richtárik, Igor Sokolov, and Ilyas Fatkhullin. 2021. EF21: A New, Simpler, Theoretically Better, and Practically Faster Error Feedback. In *NeurIPS*.

[24] Peter Richtárik, Igor Sokolov, Ilyas Fatkhullin, Elnur Gasanov, Zhize Li, and Eduard Gorbunov. 2022. 3PC: Three Point Compressors for Communication-Efficient Distributed Training and a Better Theory for Lazy Aggregation. arXiv:2202.00998 [cs.LG]

[25] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *MLSys*.

[26] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs. In *Interspeech*.

[27] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. 2011. Sharing the Data Center Network. In *NSDI*.

[28] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. 2018. Sparsified SGD with Memory. In *NeurIPS*.

[29] Ananda Theertha Suresh, Felix X. Yu, Sanjiv Kumar, and H. Brendan McMahan. 2017. Distributed Mean Estimation with Limited Communication. In *ICML*.

[30] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. 2019. PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization. In *NeurIPS*.

[31] Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary Charles, Dimitris Papailiopoulos, and Stephen Wright. 2018. Atomo: Communication-efficient learning via atomic sparsification. In *NeurIPS*.

[32] Jue Wang, Yucheng Lu, Binhang Yuan, Beidi Chen, Percy Liang, Christopher De Sa, Christopher Re, and Ce Zhang. 2023. CocktailSGD: Fine-tuning Foundation Models over 500Mbps Networks. In *ICML*.

[33] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. 2019. Adaptive Federated Learning in Resource Constrained Edge Computing Systems. *IEEE Journal on Selected Areas in Communications* 37, 6 (2019), 1205–1221.

[34] Yiding Wang, Decang Sun, Kai Chen, Fan Lai, and Mosharaf Chowdhury. 2023. Egeria: Efficient DNN Training with Knowledge-Guided Layer Freezing. In *EuroSys*.

[35] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning. In *NeurIPS*.

[36] Hang Xu, Chen-Yu Ho, Ahmed M. Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis. 2021. GRACE: A Compressed Communication Framework for Distributed Machine Learning. In *ICDCS*.

[37] Jie Xu, Heqiang Wang, and Lixing Chen. 2022. Bandwidth Allocation for Multiple Federated Learning Services in Wireless Edge Networks. *IEEE Transactions on Wireless Communications* 21, 4 (2022), 2534–2546.

A PROBLEM FORMULATION

Algorithm 2 is a generic form for solving problem (1) which can be used as a baseline.

Algorithm 2 A Generic Distributed Training Meta-Algorithm

1: **Input:** loss ℓ; weights w_m, datasets $\mathcal{D}_m$ and algorithms A_m^{update} for $m \in [M]$; learning rate schedule $\{\gamma^k\}$ for iterations $k \geq 0$
2: Initialize with model $x^0 \in \mathbb{R}^d$ on the server
3: **for** $k = 0, 1, 2, \dots$ **do**
4:　　The server broadcasts model x^k to all machines $m \in [M]$
5:　　Each machine $m \in [M]$ uses algorithm A_m to compute the update

$$u_m^k = A_m^{\text{update}}\left(x^k, \ell, \mathcal{D}_m\right) \in \mathbb{R}^d$$

6:　　Each machine $m \in [M]$ uploads the update u_m^k to the server
7:　　The server updates the model via

$$x^{k+1} = x^k - \gamma^k \sum_{m=1}^{M} w_m u_m^k,$$

　　where $\gamma^k > 0$ is a learning rate
8: **end for**

Here are some canonical examples:

- If A_m^{update} performs one step of gradient descent with respect to function f_m, i.e.,

$$u_m^k = \alpha^k \nabla \mathrm{E}_{\xi \sim \mathcal{D}_m}\left[\ell(x^k, \xi)\right] = \alpha^k \nabla f_m(x^k),$$

  where α_m^k is a step size, then Algorithm 2 becomes gradient descent (with step size $\gamma^k \alpha^k$) for solving problem (1). If A_m^{update} applies multiple steps of gradient descent instead, then Algorithm 2 becomes local gradient descent [14, 15].
- If A_m^{update} performs one step of stochastic gradient descent with respect to function f_m, i.e.,

$$u_m^k = \frac{1}{b} \sum_{j=1}^{b} \nabla \ell(x^k, \xi_j^k), \qquad \text{where} \qquad \xi_1^k, \dots, \xi_b^k \sim \mathcal{D}_m,$$

  then Algorithm 2 becomes a variant of mini-batch stochastic gradient descent for solving problem (1). If A_m^{update} applies multiple steps of stochastic gradient descent instead, then Algorithm 2 becomes local stochastic gradient descent [15].

In practice, not all workers will participate in every epoch's training. There are many worker sampling algorithms proposed [19, 25] to speed up the training. However, these algorithms can also introduce bias and have various behaviors on different tasks. In this work, we consider the situation of full participation of workers to avoid the influence of worker sampling.

(1) The server broadcasts model x^k to all workers $m \in [M]$;
(2) Each machine $m \in [M]$ computes update

$$\hat{u}_m^k = A_m\left(x^k, \ell, \mathcal{D}_m\right)$$

　　via some algorithm A_m and uploads the update to the server;
(3) The server aggregates the updates and updates the model via

$$x^{k+1} = x^k - \gamma^k \sum_{m=1}^{M} w_m \hat{u}_m^k,$$

　　where γ^k is a learning rate.

B KIMAD ALGORITHM WITH EXPLANATION

Algorithm 3 illustrates the Kimad algorithm with more details and comments.

Algorithm 3 Kimad: Adaptive Gradient Compression with Bandwidth Awareness (Detailed)

1: **Input:** loss ℓ; weights w_m, datasets $\mathcal{D}_m$ and algorithms A_m^{update} for computing the model update on each machine $m \in [M]$; set of compressors Ω; compressor-selection algorithm A^{compress} used by the server and the machines; model $x^0 \in \mathbb{R}^d$ known by the server; initial model estimator $\hat{x}^{-1} \in \mathbb{R}^d$ known by the machines and the server (for example, $\hat{x}^{-1} = 0$ or $\hat{x}^{-1} = x^0$ are acceptable choices); initial update estimators $\hat{u}_m^{-1} \in \mathbb{R}^d$ for $m \in [M]$ known by the machines and the server (for example, $\hat{u}_m^{-1} = 0$ for all $m \in [M]$ is an acceptable choice); single round time budget $t > 0$; learning rate schedule $\{\gamma^k\}$ for iterations $k \geq 0$

2: **for each communication round** $k = 0, 1, 2, \dots$ **do**

3:     The server estimates the broadcast/downlink bandwidth at communication round k; let the estimate be B^k

4:     The server chooses a compressor $C^k \in \Omega$ for compressing the difference $x^k - \hat{x}^{k-1}$ via algorithm A^{compress}

$$C^k = A^{\text{compress}}(\Omega, x^k, \hat{x}^{k-1}, B^k, t)$$

(The algorithm A^{compress} aims to choose the compressor from Ω suffering minimal error when compressing the difference $x^k - \hat{x}^{k-1}$, subject to the constraint that the compressed message should take at most t seconds to broadcast to the machines given the broadcast bandwidth estimate B^k)

5:     The server updates the model estimator to

$$\hat{x}^k = \hat{x}^{k-1} + C^k(x^k - \hat{x}^{k-1})$$

6:     The server broadcasts the compressed vector $C^k(x^k - \hat{x}^{k-1})$ to all machines $m \in [M]$

7:     **for each machine** $m = 1, 2, \dots, M$ **in parallel do**

8:         Update the model estimator to

$$\hat{x}^k = \hat{x}^{k-1} + C^k(x^k - \hat{x}^{k-1})$$

using the previously stored estimator $\hat{x}^{k-1}$ and the received message

9:         Use algorithm A_m^{update} to compute the update

$$u_m^k = A_m^{\text{update}}\left(\hat{x}^k, \ell, \mathcal{D}_m\right) \in \mathbb{R}^d$$

10:         Estimate the uplink bandwidth of machine m at communication round k; let the estimate be B_m^k

11:         Choose a compressor $C_m^k \in \Omega$ for compressing the difference $u_m^k - \hat{u}_m^{k-1}$ via algorithm A^{compress}

$$C_m^k = A^{\text{compress}}(\Omega, u_m^k, \hat{u}_m^{k-1}, B_m^k, t)$$

(The algorithm A^{compress} aims to choose the compressor from Ω suffering minimal error when compressing the difference $u_m^k - \hat{u}_m^{k-1}$, subject to the constraint that the compressed message should take at most t seconds to upload to the server given the uplink bandwidth estimate B_m^k)

12:         Upload the compressed vector $C_m^k(u_m^k - \hat{u}_m^{k-1})$ to the server

13:     **end for**

14:     The server updates all update estimators to

$$\hat{u}_m^k = \hat{u}_m^{k-1} + C_m^k(u_m^k - \hat{u}_m^{k-1}), \qquad m \in [M]$$

15:     The server updates the model via

$$x^{k+1} = x^k - \gamma^k \sum_{m=1}^{M} w_m \hat{u}_m^k,$$

where $\gamma^k > 0$ is a learning rate

16: **end for**

C KIMAD+ DYNAMIC PROGRAMMING

Algorithm 4 lists the dynamic programming algorithm to optimize the layer-wise compression ratio allocation to minimize the compression error.

Algorithm 4 A dynamic programming algorithm to allocate compression ratio across layers

---

1: **Input:** Model Layers L_i, accumulated gradients G_i, possible compression parameters $C = \{c^1, c^2, \ldots, c^k\}$, model compression budget $\mathcal{E}_{\max}$, discretization factor D
2: N = number of layers
3: Compute $\mathcal{E}_{\max}$ for the default compression parameters C_i^d
4: Compute discretization step $\mathcal{E}_{\max}/D$.
5: Costs matrix $N \times |C|$ where position i, j has a value of the size of layer i compressed with compression parameter c^j.
6: Errors matrix $N \times |C|$ where position i, j has a value of the discretized L_2 of the compression error when the accumulated gradients of layer i are compressed with parameter c^j.
7: DP matrix $N \times (D + 1)$ filled with ∞ values.
8: PD matrix $N \times (D + 1)$.
9: **for** $c \in C$ **do**
10:     $DP[1][Costs[1][c]] = Errors[1][c]$
11:     $PD[1][Costs[1][c]] = c$
12: **end for**
13: // Dynamic programming algorithm
14: **for** Layer $l_i := 2..N$ **do**
15:     **for** $c_i \in C$ **do**
16:         **for** $cost_i := Costs[l_i][c_i]..D$ **do**
17:             $t = DP[l_i - 1][cost_i - Costs[l_i][c_i]] + Errors[l_i][c_i]$
18:             **if** $t < DP[l_i][e_i]$ **then**
19:                 $DP[l_i][e_i] = t$
20:                 $PD[l_i][e_i] = c_i$
21:             **end if**
22:         **end for**
23:     **end for**
24: **end for**
25: $cost_{min} = argmin(DP[N])$
26: // Reconstruction of the optimal parameters
27: **for** $l_i = N..1$ **do**
28:     $result[l_i] = PD[l_i][cost_{min}]$
29:     $cost_{min} = cost_{min} - Costs[l_i][result[l_i]]$
30: **end for**
31: **return** $result$

D ASSUMPTIONS AND BASIC IDENTITIES

D.1 Layer Smoothness

ASSUMPTION 1 (LAYER SMOOTHNESS). *There exist constants $L_1, \ldots, L_\ell > 0$ such that*

$$f(x + s) \leq f(x) + \langle \nabla f(x), s \rangle + \frac{1}{2} \sum_{i=1}^{\ell} L_i \|s_i\|^2$$

holds for all $x \in \mathbb{R}^d$ and $s = (s_1, \ldots, s_\ell) \in \mathbb{R}^d$.

D.2 Global Smoothness

ASSUMPTION 2 (GLOBAL SMOOTHNESS). *There exists a constant $L > 0$ such that*

$$\|\nabla f(x + s) - \nabla f(x)\| \leq L \|s\| \tag{11}$$

holds for all $x, s \in \mathbb{R}^d$.

D.3 Definition of $\bar{G}^k$

Choose any $\delta_1, \ldots, \delta_\ell > 0$ and define

$$\bar{G}^k := \mathrm{E}\left[G^k\right], \; G^k := \sum_{i=1}^{\ell} \delta_i G_i^k, \; G_i^k := \left\|\hat{u}_i^k - \nabla_i f(x^k)\right\|^2. \tag{12}$$

D.4 Young's inequality

For any $x, y \in \mathbb{R}^d$ and any $\zeta > 0$ we have

$$\|x + y\|^2 \le (1 + \zeta) \|x\|^2 + (1 + \zeta^{-1}) \|y\|^2. \tag{13}$$

E TECHNICAL LEMMAS

Lemma 1 (Technical identity). *Let $x, \hat{u} \in \mathbb{R}^d$ and*

$$x^+ := x - \gamma \hat{u}, \tag{14}$$

where $\gamma > 0$. Then for any $M > 0$ and any $h \in \mathbb{R}^d$ we have the identity

$$\langle h, x^+ - x \rangle + \frac{M}{2} \|x^+ - x\|^2 = -\frac{\gamma}{2} \|h\|^2 - \left(\frac{1}{2\gamma} - \frac{M}{2} \right) \|x^+ - x\|^2 + \frac{\gamma}{2} \|\hat{u} - h\|^2. \tag{15}$$

Let $x^k = (x_1^k, \ldots, x_\ell^k)$. The lemma below holds for any algorithm of the form

$$x_i^{k+1} = x_i^k - \gamma_i^k \hat{u}_i^k, \qquad i = 1, \ldots, \ell,$$

where $\gamma_1^k, \ldots, \gamma_\ell^k > 0$ are stepsizes. Therefore, it also holds for Algorithm (5)–(7).

Lemma 2 (Descent). *If Assumption 1 holds, then*

$$f(x^{k+1}) \le f(x^k) + \sum_{i=1}^\ell \left(-\frac{\gamma_i^k}{2} \left\| \nabla_i f(x^k) \right\|^2 - \left(\frac{1}{2\gamma_i^k} - \frac{L_i}{2} \right) \left\| x_i^{k+1} - x_i^k \right\|^2 + \frac{\gamma_i^k}{2} \left\| \hat{u}_i^k - \nabla_i f(x^k) \right\|^2 \right).$$

PROOF. By applying Lemma 1 with $d \leftarrow d_i$, $x^+ \leftarrow x_i^{k+1}$, $x \leftarrow x_i^k$, $\gamma \leftarrow \gamma_i^k$, $\hat{u} \leftarrow \hat{u}_i^k$, $h \leftarrow \nabla_i f(x^k)$ and $M \leftarrow L_i$, we get

$$
\begin{aligned}
\left\langle \nabla_i f(x^k), x_i^{k+1} - x_i^k \right\rangle + \frac{L_i}{2} \left\| x_i^{k+1} - x_i^k \right\|^2 \;=\;& -\frac{\gamma_i^k}{2} \left\| \nabla_i f(x^k) \right\|^2 - \left(\frac{1}{2\gamma_i^k} - \frac{L_i}{2} \right) \left\| x_i^{k+1} - x_i^k \right\|^2 \\
& + \frac{\gamma_i^k}{2} \left\| \hat{u}_i^k - \nabla_i f(x^k) \right\|^2.
\end{aligned} \tag{16}
$$

Using Assumption 1, with $x \leftarrow x^k = (x_1^k, \ldots, x_\ell^k)$ and $s \leftarrow x^{k+1} - x^k = (x_1^{k+1} - x_1^k, \ldots, x_\ell^{k+1} - x_\ell^k) = (-\gamma_1^k \hat{u}_1^k, \ldots, -\gamma_\ell^k \hat{u}_\ell^k)$, we get

$$
\begin{aligned}
f(x^{k+1}) \;\le\;& f(x^k) + \sum_{i=1}^\ell \left\langle \nabla_i f(x^k), x_i^{k+1} - x_i^k \right\rangle + \frac{1}{2} \sum_{i=1}^\ell L_i \left\| x_i^{k+1} - x_i^k \right\|^2 \\
=\;& f(x^k) + \sum_{i=1}^\ell \left(\left\langle \nabla_i f(x^k), x_i^{k+1} - x_i^k \right\rangle + \frac{L_i}{2} \left\| x_i^{k+1} - x_i^k \right\|^2 \right) \\
\overset{(16)}{=}\;& f(x^k) + \sum_{i=1}^\ell \left(-\frac{\gamma_i^k}{2} \left\| \nabla_i f(x^k) \right\|^2 - \left(\frac{1}{2\gamma_i^k} - \frac{L_i}{2} \right) \left\| x_i^{k+1} - x_i^k \right\|^2 + \frac{\gamma_i^k}{2} \left\| \hat{u}_i^k - \nabla_i f(x^k) \right\|^2 \right).
\end{aligned}
$$

□

Our next lemma is specific to Algorithm (5)–(7).

Lemma 3 (3PC inequality). *Choose any $i \in [\ell]$. Let $C_i^k \in \mathbb{C}^{d_i}(\alpha_i)$ for $0 < \alpha_i \le 1$. Define $G_i^k := \left\| \hat{u}_i^k - \nabla_i f(x^k) \right\|^2$ and $W^k := \{\hat{u}_i^k, \ldots, \hat{u}_\ell^k, x^k, x^{k+1}\}$. Then*

$$\mathrm{E}\left[G_i^{k+1} \mid W^k \right] \le (1 - \theta_i) G_i^k + \beta_i \left\| \nabla_i f(x^{k+1}) - \nabla_i f(x^k) \right\|^2, \tag{17}$$

where

$$\theta_i := 1 - (1 - \alpha_i)(1 + \zeta_i), \qquad \text{and} \qquad \beta_i := (1 - \alpha_i) \left(1 + \zeta_i^{-1} \right) \tag{18}$$

and ζ_i is any positive number.

Proof.

$$
\begin{aligned}
\mathrm{E}\left[G_i^{k+1} \mid W^k\right] &= \mathrm{E}\left[\left\|\hat{u}_i^{k+1} - \nabla_i f(x^{k+1})\right\|^2 \mid W^k\right] \\
&\stackrel{(6)}{=} \mathrm{E}\left[\left\|\hat{u}_i^k + C_i^k\left(\nabla_i f(x^{k+1}) - \hat{u}_i^k\right) - \nabla_i f(x^{k+1})\right\|^2 \mid W^k\right] \\
&\leq (1-\alpha_i)\left\|\nabla_i f(x^{k+1}) - \hat{u}_i^k\right\|^2 \\
&= (1-\alpha_i)\left\|\nabla_i f(x^k) - \hat{u}_i^k + \nabla_i f(x^{k+1}) - \nabla_i f(x^k)\right\|^2 \\
&\stackrel{(13)}{\leq} (1-\alpha_i)(1+\zeta_i)\left\|\nabla_i f(x^k) - \hat{u}_i^k\right\|^2 + (1-\alpha_i)\left(1+\zeta_i^{-1}\right)\left\|\nabla_i f(x^{k+1}) - \nabla_i f(x^k)\right\|^2,
\end{aligned}
$$

where the first inequality holds since $C_i^k \in \mathbb{C}^{d_i}(\alpha_i)$, and in the last step we have applied Young's inequality. $\qquad \square$

F   PROOF OF THEOREM 1

Proof. We proceed in three steps:

STEP 1. First, we note that Lemma 3 says that

$$
\mathrm{E}\left[\left\|\hat{u}_i^{k+1} - \nabla_i f(x^{k+1})\right\|^2 \mid W^k\right] \stackrel{(17)}{\leq} (1-\theta_i)\left\|\hat{u}_i^k - \nabla_i f(x^k)\right\|^2 + \beta_i\left\|\nabla_i f(x^{k+1}) - \nabla_i f(x^k)\right\|^2. \tag{19}
$$

Adding inequalities (19) over $i \in [\ell]$ and recalling that $G_i^k := \left\|\hat{u}_i^k - \nabla_i f(x^k)\right\|^2$, we get

$$
\begin{aligned}
\mathrm{E}\left[G^{k+1} \mid W^k\right] &\stackrel{(12)}{=} \mathrm{E}\left[\sum_{i=1}^{\ell} \delta_i G_i^{k+1} \mid W^k\right] \\
&\stackrel{(12)}{=} \mathrm{E}\left[\sum_{i=1}^{\ell} \delta_i\left\|\hat{u}_i^{k+1} - \nabla_i f(x^{k+1})\right\|^2 \mid W^k\right] \\
&= \sum_{i=1}^{\ell} \delta_i \mathrm{E}\left[\left\|\hat{u}_i^{k+1} - \nabla_i f(x^{k+1})\right\|^2 \mid W^k\right] \\
&\stackrel{(19)}{\leq} \sum_{i=1}^{\ell} \delta_i\left((1-\theta_i)\left\|\hat{u}_i^k - \nabla_i f(x^k)\right\|^2 + \beta_i\left\|\nabla_i f(x^{k+1}) - \nabla_i f(x^k)\right\|^2\right) \\
&= \sum_{i=1}^{\ell} \delta_i (1-\theta_i)\left\|\hat{u}_i^k - \nabla_i f(x^k)\right\|^2 + \sum_{i=1}^{\ell} \delta_i \beta_i\left\|\nabla_i f(x^{k+1}) - \nabla_i f(x^k)\right\|^2 \\
&\stackrel{(12)}{=} \left(1-\min_i \theta_i\right) G^k + \sum_{i=1}^{\ell} \delta_i \beta_i\left\|\nabla_i f(x^{k+1}) - \nabla_i f(x^k)\right\|^2 \\
&\leq \left(1-\min_i \theta_i\right) G^k + \left(\max_i \delta_i \beta_i\right)\sum_{i=1}^{\ell}\left\|\nabla_i f(x^{k+1}) - \nabla_i f(x^k)\right\|^2 \\
&= \left(1-\min_i \theta_i\right) G^k + \left(\max_i \delta_i \beta_i\right)\left\|\nabla f(x^{k+1}) - \nabla f(x^k)\right\|^2 \\
&\stackrel{(11)}{\leq} \left(1-\min_i \theta_i\right) G^k + \left(\max_i \delta_i \beta_i\right) L^2\left\|x^{k+1} - x^k\right\|^2. \tag{20}
\end{aligned}
$$

Using the Tower property in (20), we proceed to

$$
\mathrm{E}\left[G^{k+1}\right] = \mathrm{E}\left[\mathrm{E}\left[G^{k+1} \mid W^k\right]\right] \stackrel{(20)}{\leq} (1-\theta)\mathrm{E}\left[G^k\right] + \left(\max_i \delta_i \beta_i\right) L^2 \mathrm{E}\left[\left\|x^{k+1} - x^k\right\|^2\right]. \tag{21}
$$

STEP 2. Next, using Lemma 2, we obtain the bound

$$
\begin{aligned}
f(x^{k+1}) &\leq f(x^k) + \sum_{i=1}^{\ell}\left(-\frac{\gamma_i^k}{2}\left\|\nabla_i f(x^k)\right\|^2 - \left(\frac{1}{2\gamma_i^k} - \frac{L_i}{2}\right)\left\|x_i^{k+1} - x_i^k\right\|^2 + \frac{\gamma_i^k}{2}\left\|\hat{u}_i^k - \nabla_i f(x^k)\right\|^2\right). \\
&= f(x^k) - \sum_{i=1}^{\ell}\frac{\gamma_i^k}{2}\left\|\nabla_i f(x^k)\right\|^2 - \sum_{i=1}^{\ell}\left(\frac{1}{2\gamma_i^k} - \frac{L_i}{2}\right)\left\|x_i^{k+1} - x_i^k\right\|^2 \\
&\quad + \sum_{i=1}^{\ell}\frac{\gamma_i^k}{2\delta_i}\delta_i\left\|\hat{u}_i^k - \nabla_i f(x^k)\right\|^2 \\
&\overset{(12)}{\leq} f(x^k) - \sum_{i=1}^{\ell}\frac{\gamma_i^k}{2}\left\|\nabla_i f(x^k)\right\|^2 - \sum_{i=1}^{\ell}\left(\frac{1}{2\gamma_i^k} - \frac{L_i}{2}\right)\left\|x_i^{k+1} - x_i^k\right\|^2 + \left(\max_i\frac{\gamma_i^k}{2\delta_i}\right)G^k. \qquad (22)
\end{aligned}
$$

Subtracting f^{inf} from both sides of (22) and taking expectation, we get

$$
\begin{aligned}
\mathrm{E}\left[f(x^{k+1}) - f^{\text{inf}}\right] &\leq \mathrm{E}\left[f(x^k) - f^{\text{inf}}\right] - \sum_{i=1}^{\ell}\frac{\gamma_i^k}{2}\mathrm{E}\left[\left\|\nabla_i f(x^k)\right\|^2\right] \\
&\quad - \sum_{i=1}^{\ell}\left(\frac{1}{2\gamma_i^k} - \frac{L_i}{2}\right)\mathrm{E}\left[\left\|x_i^{k+1} - x_i^k\right\|^2\right] + \left(\max_i\frac{\gamma_i^k}{2\delta_i}\right)\mathrm{E}\left[G^k\right] \\
&= \mathrm{E}\left[f(x^k) - f^{\text{inf}}\right] - \frac{\gamma}{2}\sum_{i=1}^{\ell}w_i\mathrm{E}\left[\left\|\nabla_i f(x^k)\right\|^2\right] \\
&\quad - \sum_{i=1}^{\ell}\left(\frac{1}{2\gamma w_i} - \frac{L_i}{2}\right)\mathrm{E}\left[\left\|x_i^{k+1} - x_i^k\right\|^2\right] + \frac{\gamma}{2}\left(\max_i\frac{w_i}{\delta_i}\right)\mathrm{E}\left[G^k\right], \qquad (23)
\end{aligned}
$$

where we used the fact that $\gamma_i^k \equiv \gamma w_i$.

STEP 3: COMBINING PREVIOUS STEPS.. Due to the restriction $(1 - \alpha_i)(1 + \zeta_i) < 1$, which holds for all $i \in [\ell]$, we know that $\theta_i > 0$ for all i, and therefore, $\theta = \min_i \theta_i$ is also positive. This detail plays a role in what follows.

Let $\Delta^k := \mathrm{E}\left[f(x^k) - f^{\text{inf}}\right]$, $\bar{G}^k := \mathrm{E}\left[G^k\right]$, $R_i^k := \mathrm{E}\left[\left\|x_i^{k+1} - x_i^k\right\|^2\right]$ and $R^k := \sum_{i=1}^{\ell}R_i^k = \mathrm{E}\left[\left\|x^{k+1} - x^k\right\|^2\right]$. By adding (23) to the $\gamma\frac{\left(\max_i\frac{w_i}{\delta_i}\right)}{2\theta}$ multiple of (21), we obtain

$$
\begin{aligned}
\Delta^{k+1} + \frac{\gamma\left(\max_i\frac{w_i}{\delta_i}\right)}{2\theta}\bar{G}^{k+1} &\leq \Delta^k - \frac{\gamma}{2}\sum_{i=1}^{\ell}w_i\mathrm{E}\left[\left\|\nabla_i f(x^k)\right\|^2\right] - \sum_{i=1}^{\ell}\left(\frac{1}{2\gamma w_i} - \frac{L_i}{2}\right)R_i^k \\
&\quad + \frac{\gamma}{2}\left(\max_i\frac{w_i}{\delta_i}\right)\bar{G}^k + \frac{\gamma\left(\max_i\frac{w_i}{\delta_i}\right)}{2\theta}\left((1 - \theta)\bar{G}^k + \left(\max_i\delta_i\beta_i\right)L^2 R^k\right) \\
&= \Delta^k + \frac{\gamma\left(\max_i\frac{w_i}{\delta_i}\right)}{2\theta}\bar{G}^k - \frac{\gamma}{2}\sum_{i=1}^{\ell}w_i\mathrm{E}\left[\left\|\nabla_i f(x^k)\right\|^2\right] \\
&\quad - \sum_{i=1}^{\ell}\left(\frac{1}{2\gamma w_i} - \frac{L_i}{2} - \frac{\gamma\left(\max_i\frac{w_i}{\delta_i}\right)}{2\theta}\left(\max_i\delta_i\beta_i\right)L^2\right)R_i^k \\
&\leq \Delta^k + \frac{\gamma\left(\max_i\frac{w_i}{\delta_i}\right)}{2\theta}\bar{G}^k - \frac{\gamma}{2}\sum_{i=1}^{\ell}w_i\mathrm{E}\left[\left\|\nabla_i f(x^k)\right\|^2\right].
\end{aligned}
$$

The last inequality follows from the bound (9). By summing up inequalities for $k = 0, \ldots, K - 1$, we get

$$
0 \leq \Delta^K + \frac{\gamma\left(\max_i\frac{w_i}{\delta_i}\right)}{2\theta}\bar{G}^K \leq \Delta^0 + \frac{\gamma\left(\max_i\frac{w_i}{\delta_i}\right)}{2\theta}\bar{G}^0 - \frac{\gamma}{2}\sum_{k=0}^{K-1}\left(\sum_{i=1}^{\ell}w_i\mathrm{E}\left[\left\|\nabla_i f(x^k)\right\|^2\right]\right).
$$

Jihao Xin, Ivan Ilin, Shunkang Zhang, Marco Canini, and Peter Richtárik

Multiplying both sides by $\frac{2}{\gamma K}$, after rearranging we get

$$
\frac{1}{K} \sum_{k=0}^{K-1} \left(\sum_{i=1}^{\ell} w_i \mathrm{E} \left[\left\| \nabla_i f(x^k) \right\|^2 \right] \right) \leq \frac{2\Delta^0}{\gamma K} + \frac{\left(\max_i \frac{w_i}{\delta_i} \right) \bar{G}^0}{\theta K} .
$$

□