

# Journal of Parallel and Distributed Computing

## ARGO: Overcoming Hardware Dependence in Distributed Learning

--Manuscript Draft--

<b>Manuscript Number:</b>	JPDC-D-23-00450
<b>Article Type:</b>	VSI:SBAC-PAD 2022 [MGE-Cristina Boeres]
<b>Section/Category:</b>	Special Issue Papers
<b>Keywords:</b>	Distributed learning; Hardware heterogeneity; Byzantine resilience; Stochastic optimization
<b>Manuscript Region of Origin:</b>	Africa
<b>Abstract:</b>	<p>Mobile devices offer a valuable resource for distributed learning alongside traditional computers, encouraging energy efficiency and privacy through local computations. However, the hardware limitations of these devices makes it impossible to use classical SGD for industry-grade machine learning models (with a very large number of parameters). Moreover, they are intermittently available and susceptible to failures. To address these challenges, we introduce ARGO, an algorithm that combines adaptive workload schemes with Byzantine resilience mechanisms, as well as dynamic device participation. Our theoretical analysis demonstrates linear convergence for strongly convex losses and sub-linear convergence for non-convex losses, without assuming specific dataset partitioning (for potential data heterogeneity). Our formal analysis highlights the interplay between convergence properties, hardware capabilities, Byzantine impact, and standard factors such as mini-batch size and learning rate. Through extensive evaluations, we show that ARGO outperforms standard SGD in terms of convergence speed and accuracy, and, most importantly, thrives when classical SGD is not possible due to hardware limitations.</p>

# ARGO: Overcoming Hardware Dependence in Distributed Learning

Karim Boubouh<sup>a</sup>, Amine Boussetta<sup>a,\*</sup>, Rachid Guerraoui<sup>b</sup>, Alexandre  
Maurer<sup>a</sup>

<sup>a</sup>*College of Computing, UM6P, Benguerir, Morocco*

<sup>b</sup>*Distributed Computing Laboratory, EPFL, Lausanne, Switzerland*

---

\*Corresponding author

*Email addresses:* `karim.boubouh@um6p.ma` (Karim Boubouh),  
`amine.boussetta@um6p.ma` (Amine Boussetta), `rachid.guerraoui@epfl.ch` (Rachid  
Guerraoui), `alexandre.maurer@um6p.ma` (Alexandre Maurer)

*Preprint submitted to Journal of Parallel and Distributed Computing*

*May 25, 2023*

## Abstract

Mobile devices offer a valuable resource for distributed learning alongside traditional computers, encouraging energy efficiency and privacy through local computations. However, the hardware limitations of these devices makes it impossible to use classical SGD for industry-grade machine learning models (with a very large number of parameters). Moreover, they are intermittently available and susceptible to failures. To address these challenges, we introduce ARGO, an algorithm that combines adaptive workload schemes with Byzantine resilience mechanisms, as well as dynamic device participation. Our theoretical analysis demonstrates linear convergence for strongly convex losses and sub-linear convergence for non-convex losses, without assuming specific dataset partitioning (for potential data heterogeneity). Our formal analysis highlights the interplay between convergence properties, hardware capabilities, Byzantine impact, and standard factors such as mini-batch size and learning rate. Through extensive evaluations, we show that ARGO outperforms standard SGD in terms of convergence speed and accuracy, and, most importantly, thrives when classical SGD is not possible due to hardware limitations.

**Note for reviewers:** this paper is the journal version of "Democratizing machine learning: Resilient distributed learning with heterogeneous participants", recently published in the 41st International Symposium on Reliable Distributed Systems (SRDS 2022). For your information, the changes and new results are explained in Section 1 of the supplementary material.

*Keywords:* Distributed learning, Hardware heterogeneity, Byzantine resilience, Stochastic optimization

## 1. Introduction

Neural processing has been at the core of artificially intelligent applications in domains such as computer vision, speech recognition and natural language processing. Overparameterization [1] (i.e. having more model parameters than data points) was shown to have a positive impact on the performance of machine learning (ML) models, and therefore, the outstanding performance witnessed today is mostly due to the increasing size of these models. As a matter of fact, AlexNet [2], one of the early neural network architectures that has been proposed for the ImageNet classification challenge [3] with 1.2 million data points, has 60 million parameters, while recent architectures like VGG16 [4] and CoCa [5] have 138 million and 2.1 billion parameters, respectively. Perhaps the strongest example of size lies in the category of large language models: ChatGPT [6], an AI-enabled conversational agent, has more than 175 billion model parameters. However, training these models face criticism regarding energy consumption, privacy, and security.

To address (at least) the problem of energy consumption, one may be tempted to train these large models by leveraging one of the biggest networks of electronic devices in the world: smartphones. In fact, the number of mobile device users is forecast to exceed 7.8 billion by 2028, representing more than 90% of the estimated world’s population.<sup>1</sup>, and these devices are considered nowadays as the main source of data generation. If this huge network of devices is exploited under the Federated Learning [7] scheme, thus solving

---

<sup>1</sup>According to the Mobile Economy 2019 report by the GSM Association: <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide>

privacy concerns (to some extent), the resulting setting seems ready to host large models training. Sadly, the aforementioned sentence is not true.

The primary limitation when using mobile devices is their constrained hardware capabilities. Even if a large model can fit into a smartphone’s memory, accessing it requires an excessive amount of energy that surpasses the device’s power limits, as documented in [8]. Additionally, smartphones are not consistently accessible. They may disconnect from networks, run out of battery, or lack free resources at specific times of the day. Furthermore, smartphones can encounter failures due to software or hardware issues, and their behavior can be arbitrary, since they are owned by untrusted individuals. Essentially, a smartphone can exhibit Byzantine<sup>2</sup> characteristics [9]. At this juncture, one might wonder: is it feasible to address the challenges of hardware heterogeneity, availability, energy efficiency and Byzantine resilience, all at once?

We answer positively by proposing ARGO (*Atomic Resilient Gradient-based Optimization*), a new algorithm combining our adaptive workload technique (i.e. allowing partial computation via partial derivatives or reduced parameter vectors) with established Byzantine resilience mechanisms. The algorithm also allows dynamic participation of devices, as long as a quorum of workers is available (a quorum which size depends on the upper bound of the number of Byzantine workers). We analyze our algorithm without assuming any specific data partitioning (due to possible data heterogeneity), and including biased estimation (which makes our formal analysis compati-

---

<sup>2</sup>The Byzantine abstraction is commonly used to model all kind of failures (e.g. crash, omission, software, hardware or temporal failures, as well as adversarial attacks).

ble with most gradient compression techniques, for instance). Theoretically, we prove the convergence of ARGO, and establish for the first time a direct link between Byzantine resilience and hardware heterogeneity. Empirically, we show that our protocol outperforms the standard learning scheme on both accuracy and execution time on different classification tasks. We also analyze the energy consumption of our algorithm (executed on smartphones) versus the classical SGD (executed in a server machine), and show that training ML models on mobile devices using our algorithm is energy efficient.

## 2. Related Work

Previous distributed ML works address heterogeneous computational capabilities and Byzantine behavior separately. Some works [10, 11, 12, 13, 14, 15, 16, 17] focus on Byzantine resilience using robust statistics, filtering schemes, or coding theory. However, they assume homogeneous data distribution and computational capabilities.

Others [12, 18, 19] claiming computational heterogeneity awareness mainly consider asynchrony, assuming that workers can compute a full gradient but may be stale. They propose synchronization schedules or adaptive learning rate schedules (or both) to tackle straggler issues. Gradient compression schemes [20, 21, 22, 23] reduce communication complexity but do not address computational heterogeneity, since workers still compute the full-sized model gradient.

The closest related works, [24, 25] and [26], explore partial computation but do not analyze the impact of hardware capabilities on convergence. They also overlook Byzantine failures and assume homogeneous datasets, with con-

stantly available workers.

Our algorithm operates under all these constraints concurrently. It provides deeper theoretical guarantees, including standard factors like learning rate and mini-batch size, as well as new factors like worker computation rates, Byzantine impact, server waiting time, and number of available workers selected. In this work, we introduce a trade-off between worker capabilities and Byzantine resilience.

### 3. Preliminaries

We use the parameter-server architecture [27] with a server coordinating  $n$  workers for training a machine learning model. The dataset consists of  $N$  data points  $X = x_1, \dots, x_N$ , divided among the workers. Each worker  $i$  has a set of data points denoted by  $\mathcal{S}_i$ , where  $\bigcup_{i=1}^n \mathcal{S}_i = X$ , and  $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$  for all  $i \neq j$ . The server follows the algorithm correctly, but some workers may be faulty, and their identity is unknown [9]. There are two types of workers: 1) Correct workers, represented by the set  $\mathcal{C} \subseteq [n]$ , who follow instructions accurately, and 2) Byzantine workers, represented by  $[n] \setminus \mathcal{C}$ , who can behave arbitrarily, causing crashes or injecting adversarial perturbations (see, e.g., [10]).

Our goal is to design a distributed learning algorithm that allows all correct workers to learn a model over the union of their data points  $X_{\mathcal{C}} = \bigcup_{i \in \mathcal{C}} \mathcal{S}_i$ , despite the presence of up to  $f$  Byzantine workers. Specifically, we fix a learning model parameterized by  $w \in \mathbb{R}^d$ , for which each data point  $x$  incurs a loss of  $\ell(w, x)$ , where the function  $\ell : \mathbb{R}^d \times X \rightarrow \mathbb{R}$  is assumed to be differentiable. The algorithm aims to find a local minimum of the following

function  $\mathcal{L}(w) = \frac{1}{|X_C|} \sum_{x \in X_C} \ell(w, x)$ .

Recall that in standard distributed SGD, each worker  $i$  samples a random *mini-batch*  $\zeta_{i_k} \subset \mathcal{S}_i$  of size  $s_i$  to compute its *stochastic* gradient:

$$G_i(w_k) = \frac{1}{s_i} \sum_{x \in \zeta_{i_k}} \nabla \ell(w_k, x) \quad (1)$$

Then, the workers send back their stochastic gradients to the server, which updates the model parameters as follows:  $w_{k+1} = w_k - \gamma \left( \frac{1}{n} \sum_{i=1}^n G_i(w_k) \right)$  where  $\gamma > 0$  is the learning rate.

Besides the presence of faulty workers, another challenge we consider in our model is the heterogeneous computational capabilities of the workers. We formally discuss this issue in the following subsection.

### 3.1. Device heterogeneity: computational power and data

In our work, we consider the scenario where workers cannot use the full  $d$ -dimensional model  $w = [w_1, \dots, w_d]^T$ . Instead, each worker  $i$  utilizes a sub-model  $z_i = \mathcal{M}(w)$ , where  $\mathcal{M} : \mathbb{R}^d \rightarrow \mathbb{R}^d$  is a masking scheme obtained by randomly selecting  $b_i$  elements from  $w$  and deactivating the rest. The size of the sub-model, denoted by  $b_i$ , is determined by the hardware capabilities of the worker, encompassing factors such as processing power, storage, and connection bandwidth. For conciseness, we combine these factors into a single variable called the computation rate  $\lambda_i$ , representing the size of the sub-model that can be used to compute a gradient using one data point  $x \in \mathcal{S}_i$  in a single unit of time.

Since the server allows  $\tau$  units of time to receive the responses from workers after broadcasting the actual parameter vector  $w_k$  at time  $k$ , and



each worker uses a mini batch of size  $s_i$  to compute the gradients, the size of the sub-model selected by worker  $i$  can be expressed as follows:

$$b_i = \min \left\{ \frac{\lambda_i \tau}{s_i}, d \right\} \quad (2)$$

**Bounding the sub-model size.** When the mini-batch size  $s_i$  is very low, or when the computation rate  $\lambda_i$  is very high for a worker  $i$ , the quantity  $\frac{\lambda_i \tau}{s_i}$  may exceed the full model size  $d$ . There is a trivial upper bound of  $d$  for this quantity. However, there also exists a non-trivial lower bound based on the ML model. For example, in a linear regression model, the minimum number of weights required to make a prediction is  $b_{min} = 1$ . In a fully-connected feed-forward neural network, the minimum sub-network size is determined by the number of inputs, hidden layers, and outputs. The server must set a waiting time  $\tau$  that allows the weakest worker to compute a gradient on the minimal reduced vector using  $b_{min}$  parameters. Thus, we obtain the following lower bounds for every  $i \in [n], \tau > 0$ :

$$b_i \geq b_{min} \quad \text{and} \quad \tau \geq \tau_{weak} \quad (3)$$

Assuming practitioners follow established guidelines [28], we can consider mini-batch sizes  $s_i$  to follow a normal distribution. Similarly, we can assume that hardware capabilities and computation rates  $\lambda_i$  also follow a normal distribution. Based on these assumptions, Lemma 1 provides a lower bound on the mean value of the  $k^{th}$  biggest sub-model size, among (up to)  $n$  different sizes used by the workers.

**Lemma 1.** *Let  $\lambda_i$  and  $s_i$  be the computation rate and the mini-batch size of*

worker  $i$  for  $i \in [n]$ , and  $\tau > 0$  be the server's waiting time. Suppose that  $\lambda \sim \mathcal{N}(\mu_\lambda, \sigma_\lambda^2)$  and  $s \sim \mathcal{N}(\mu_s, \sigma_s^2)$  with coefficients of variation satisfying  $\delta_\lambda = \frac{\sigma_\lambda}{\mu_\lambda} < 1$  and  $\delta_s = \frac{\sigma_s}{\mu_s} < 1$ , and  $\frac{\lambda}{s} \in \left[ \frac{\mu_\lambda}{\mu_s} \pm \left( \frac{\mu_\lambda}{\mu_s} \right) \sqrt{\delta_\lambda^2 + \delta_s^2} \right]$ . Then, the  $k^{\text{th}}$  order statistic of  $n$  independently drawn samples from the random variable  $b = \frac{\lambda\tau}{s}$  is approximated by a normal distribution, and we have:

$$\tau \frac{\hat{\mu}_\lambda}{\hat{\mu}_s} \left( 1 - \sqrt{(\delta_\lambda^2 + \delta_s^2) \frac{n-k}{k}} \right) \leq \mathbb{E}[b_{(k)}]$$

where  $\hat{\mu}_\lambda = \frac{1}{n} \sum_{i \in [n]} \lambda_i$ ,  $\hat{\mu}_s = \frac{1}{n} \sum_{i \in [n]} s_i$ ,  $\hat{\sigma}_\lambda^2 = \frac{1}{n} \sum_{i \in [n]} (\lambda_i - \mu_\lambda)^2$ ,  $\hat{\sigma}_s^2 = \frac{1}{n} \sum_{i \in [n]} (s_i - \mu_s)^2$  and  $\delta_x = \frac{\sigma_x}{\mu_x}$  for  $x \in [\lambda, s]$ .

*Proof.* (Sketch) We first estimate the location and scale parameters of the normal distributions of  $\lambda$  and  $s$  using the sample mean and sample variance:  $(\hat{\mu}_\lambda, \hat{\sigma}_\lambda)$  and  $(\hat{\mu}_s, \hat{\sigma}_s)$ . Since  $b = \frac{\lambda\tau}{s}$ , which is a fraction of two random variables multiplied by a scalar, we can approximate the ratio distribution of  $b$  with a normal distribution under some conditions on the coefficients of variance, as in [29], and we thus obtain :  $b \sim \mathcal{N}(\mu_b, \sigma_b^2)$ . Finally, using results from order statistics in [30], we have the following lower bound:

$$\mu_b - \sigma_b \sqrt{\frac{n-k}{k}} \leq \mathbb{E}[\mu_{b_{(k)}}]$$

which concludes the proof. □

### 3.2. Sub-model selection via masking schemes

In Equation 1, each worker computes a stochastic gradient of the loss function in the standard distributed SGD optimization algorithm. Our goal

is to ensure that each device can afford an SGD iteration in terms of hardware requirements. One approach is to compute only a few partial derivatives [31, 32, 33, 34], using (block) coordinate descent algorithms. However, this requires the loss function to be separable, meaning that it can be expressed as a sum or multiplication of individual univariate functions  $f_i(x_i)$ , where  $x = (x_1, \dots, x_d)$ . Computing partial derivatives is straightforward in such cases, and examples include ML models like linear regression.

For non-separable functions like artificial neural networks, the approach of computing partial derivatives alone, which reduces computation complexity at the worker’s level, is not effective. In automatic differentiation [35], most specifically the backpropagation algorithm [36], partial derivatives of complex functions are computed by applying the chain rule to their elementary components. In this framework, computing a single partial derivative requires the computation of other dependent partial derivatives. As a result, the gains in computation complexity are not as significant as in the separable case.

A solution to extend the concept of partial computation to non separable functions is to select a sub-model with a reduced parameter vector dimension, then compute the full gradient of the loss function using the reduced parameter vector. Formally, each worker applies a masking scheme  $\mathcal{M} : \mathbb{R}^d \rightarrow \mathbb{R}^d$  on the parameter vector  $w_k$  at iteration  $k$ , and we have:  $z_k = \mathcal{M}(w_k)$ .

It should be noted that, while some masking functions return unbiased estimators of the full parameter vector  $w$  (for instance the masking function in [26, 25], meaning that  $\mathbb{E}_{\mathcal{M}}[\mathcal{M}(w)] = w$ , the partial derivatives of the loss function evaluated at the reduced parameter vector  $z = \mathcal{M}(w)$  are not

necessarily unbiased estimators of the partial derivatives evaluated at  $w$ . More generally, without additional assumptions, properties on functions do not transfer to their gradients.

For generalization purposes, we use a general masking scheme  $\mathcal{M}$  with some properties on the reduced partial derivatives to conduct our theoretical analysis, as illustrated in Definition 1.

**Definition 1.** *Let  $w$  be the current parameter vector of a gradient-based optimization algorithm. Let  $\mathcal{B}$  be the set of weight indices that are active after applying a masking function  $\mathcal{M} : \mathbb{R}^d \rightarrow \mathbb{R}^d$  and let  $z = \mathcal{M}(w)$  be the reduced parameter vector.  $\mathcal{M}$  is said to be a controlled masking if there exists positive constants  $\epsilon, M'$  and  $N'$  such that, for every  $w \in \mathbb{R}^d$ ,  $x \in X_{\mathcal{C}}$  and  $j \in \mathcal{B}$ :*

$$\|\mathbb{E}_{\mathcal{M}}[\{\nabla \ell(z, x)\}_j] - \{\nabla \ell(w, x)\}_j\| \leq \epsilon$$

$$\mathbb{E}_{\mathcal{M}}[\|\{\nabla \ell(z, x)\}_j - \mathbb{E}_{\mathcal{M}}[\{\nabla \ell(z, x)\}_j]\|^2] \leq M' + N' \|\{\nabla \ell(w, x)\}_j\|^2$$

### 3.3. Robust aggregation

In the presence of Byzantine workers, the server cannot rely on the simple averaging of all workers' gradients. It should rather use a coordinate-wise robust aggregation rule  $\text{AR}$ , which we formally define below. In the following, for any vector  $u \in \mathbb{R}^d$ , its  $j$ -th coordinate is denoted by  $\{u\}_j$ .

**Definition 2.** *Let  $X_i \in \mathbb{R}^d$  be workers  $i$ 's estimate of  $\nabla \mathcal{L}(w)$  for a given  $\rho \in [n]$  and  $w \in \mathbb{R}^d$ . A real-valued aggregation rule  $\text{AR}$  is said to be  $\Delta(\rho, f)$ -robust if, for every subset  $\mathcal{Q} \subseteq [n]$  with  $|\mathcal{Q}| \geq \rho$  and for every  $j \in [d]$ , we have*

$$\mathbb{E} \left[ |\{\text{AG}_w\}_j - \{\nabla \mathcal{L}(w)\}_j|^2 \right] \leq \Delta(\rho, f) \left( \frac{1}{|\mathcal{C} \cap \mathcal{Q}|} \sum_{i \in \mathcal{C} \cap \mathcal{Q}} \sigma_{ij}^2(w) \right)$$

where  $\{\text{AG}_w\}_j = \text{AR}(\{X_i\}_j; i \in \mathcal{Q})$  and  $\sigma_{ij}^2(w) = \mathbb{E} [|\{X_i\}_j - \{\nabla \mathcal{L}(w)\}_j|^2]$ . The

robustness coefficient  $\Delta(\rho, f)$  is a bounded positive real value that depends on  $\rho$  and  $f$ .

Our theoretical analysis applies for all aggregation rules satisfying Definition 2, as soon as the robustness coefficient  $\Delta(\rho, f)$  is sufficiently well behaved: simply put, for a fixed number of Byzantine workers  $f$ , the aggregation error should at least be constant, if not decreasing, when the total number of workers  $\rho$  grows. This includes a large range of aggregation rules such as *Phocas* [37], *Trimmed Mean* [13] or *AKSEL* [11].

### 3.4. Assumptions

To formally analyze the convergence of our algorithm, we make the following standard assumption on the smoothness of the loss function [38]. We recall that  $\{u\}_j$  denotes the  $j$ -th element of vector  $u$ .

**Assumption 1** (Smoothness). *For all  $j \in [d]$ , there exists  $L_j < \infty$  such that, for all  $w, w' \in \mathbb{R}^d$ :  $|\{\nabla \mathcal{L}(w')\}_j - \{\nabla \mathcal{L}(w)\}_j| \leq L_j \|w' - w\|$*

Although most of our results do not rely on the convexity assumption, we also present a convergence result when the loss function is strongly convex, as stated below.

**Assumption 2** (Strong convexity). *There exists  $0 < \mu < \infty$  such that, for all  $w, w' \in \mathbb{R}^d$ :  $\mathcal{L}(w') \geq \mathcal{L}(w) + \langle \nabla \mathcal{L}(w), w' - w \rangle + \frac{\mu}{2} \|w' - w\|^2$  where  $\langle \cdot, \cdot \rangle$  denotes the inner product.*

To analyze the convergence of ARGO under data heterogeneity, we also make the following two assumptions. Note that we do not rely on the (stronger) assumption of uniformly bounded variance, which is indeed quite

difficult (and perhaps even impossible) to satisfy in the case of strong convexity [39]. Instead, we follow [38] and assume a non-uniform bound, as stated below in Assumption 4.

**Assumption 3** (Bounded bias). *For all  $i \in [n]$  and  $j \in [d]$ , there exist  $\delta_{i,j} < \infty$  such that, for all  $w \in \mathbb{R}^d$ :  $|\mathbb{E}_{x \sim \mathcal{S}_i} [\{\nabla \ell(w, x)\}_j] - \{\nabla \mathcal{L}(w)\}_j| \leq \delta_{i,j}$  where  $x \sim \mathcal{S}_i$  denotes a uniform sampling of  $x$  in  $\mathcal{S}_i$ .*

**Assumption 4** (Non-uniform variance). *For all  $w \in \mathbb{R}^d$ , there exists  $M_w < \infty$  and  $Q_w < \infty$  such that, for all  $i \in [n]$  and  $j \in [d]$ :*

$$\mathbb{E}_{x \sim \mathcal{S}_i} [(\{\nabla \ell(w, x)\}_j - \mathbb{E}_{x \sim \mathcal{S}_i} [\{\nabla \ell(w, x)\}_j])^2] \leq M_w + Q_w \|\{\nabla \mathcal{L}(w)\}_j\|^2$$

## 4. Atomic Resilient Gradient Descent

We now present our protocol called ARGO. *Atomic* stands for the fact that models can be reduced to match the weakest element of the network. *Resilient* represents the robustness against failures and heterogeneity.

### 4.1. Overview of the protocol

The protocol, summarized in Algorithm 1, follows  $K$  steps of the following iterative procedure.

1. At each iteration, the server maintains a parameter vector  $w_k$  and selects a consistent random order of weight indices  $C_k$ . It then broadcasts  $(w_k, C_k)$  to a random set of  $q$  workers and waits for a specific time period.
2. Upon receiving the broadcast message, an honest worker  $i$  samples a mini-batch of size  $s_i$  from its local data. It evaluates the sub-model size

$b_i$  based on its computation rate and uses a masking scheme to construct the sub-model  $z_k$ . The worker computes its stochastic gradient  $G_i(z_k)$ .

3. After the waiting period, the server identifies the set of indices  $\mathcal{B}_k$  that have been computed by at least  $v$  out of the  $q$  selected workers. It applies an aggregation rule on the reduced gradients sent by the workers, and uses the output to update the parameter vector  $w_k$ .

The details of the aggregation rule and the procedure for identifying the consistent indices are described in the algorithm.

#### 4.2. Time complexity.

While standard distributed SGD aggregates  $d$ -dimensional gradients received from  $n$  worker, and therefore waits  $\tau_s$  time units for the slowest worker, ARGO only aggregates reduced gradients from only a subset  $q$  of workers, making it faster in that sense. However, identifying appropriate coordinates and workers (steps 8, 9, 10 and 11 in Algorithm 1) introduces computation overhead, affecting overall time complexity. Thus, ARGO's speed compared to SGD depends on  $\tau_s - \tau$  and the overhead cost, approximated as  $\mathcal{O}(q \log(q) \frac{\bar{\lambda}\tau}{\bar{s}})$ . Here,  $\bar{\lambda}$  is average computation rate and  $\bar{s}$  is average mini-batch size.

## 5. Theoretical Guarantees

Inspired by prior works in the homogeneous setting [38, 10, 11, 13], we show that our algorithm eventually reaches a ball centered at a local optimum, whose radius is a function of the statistical error. In the next sections,

---

**Algorithm 1** ARGO: Atomic Resilient Gradient-based Optimization

---

- 1: **Input:**  $q$ , AR (aggregation rule),  $w_1$ ,  $v$ ,  $K$ ,  $\gamma$
- 2: **Execute the following instructions for each step**  $k = 1, \dots, K$
- 3:  $C_k \leftarrow$  consistent random permutation of  $\{1, \dots, d\}$
- 4: Select  $q$  random workers  $\{i_1, \dots, i_q\} \subseteq \{1, \dots, n\}$
- 5: Broadcast  $(w_k, C_k)$  to the selected  $q$  workers
- 6: Wait  $\tau$  time units to receive workers' gradients
- 7:  $\nabla_{i_1}, \dots, \nabla_{i_q} \leftarrow$  partial derivatives sent by the  $q$  workers
- 8:  $R \leftarrow$  sort  $(|\nabla_{i_1}|, \dots, |\nabla_{i_q}|, \text{ decreasing order})$
- 9:  $b_v \leftarrow v^{\text{th}}$  item in the list  $R$
- 10:  $\mathcal{B}_k \leftarrow$  first  $b_v$  elements in  $C_k$
- 11:  $\forall j \in \mathcal{B}_k$ , identify the set  $\mathcal{Q}_j \subseteq \{i_1, \dots, i_q\}$  of workers that sent the  $j$ -th coordinate of their reduced gradients.
- 12: Compute the aggregate of workers' partial derivatives  $\text{AG}_{w_k} \in \mathbb{R}^d$  such that, for all  $j \in [d]$ ,

$$\{\text{AG}_{w_k}\}_j = \begin{cases} \text{AR}(\{\nabla_i\}_j, i \in \mathcal{Q}_j) & \text{if } j \in \mathcal{B}_k \\ 0 & \text{otherwise} \end{cases}$$

where  $\{\nabla_i\}_j$  is sent by worker  $i$  for the  $j$ -th element of its reduced gradient.

- 13: Update the parameter vector:  $w_{k+1} = w_k - \gamma \text{AG}_{w_k}$
  - 14: **Input:** local dataset  $\mathcal{S}_i$ , mini-batch size  $s_i$ , computational capability  $\lambda_i$
  - 15: **If** the broadcast message  $(w_k, C_k)$  is received **then** execute the following steps:
  - 16: Sample a random mini-batch  $\zeta_{i_k}$  of size  $s_i$  from dataset  $\mathcal{S}_i$
  - 17:  $\mathcal{B}_k^i \leftarrow$  first  $b_i$  elements in  $C_k$ , where  $b_i = \min\{\frac{\tau\lambda_i}{s_i}, d\}$
  - 18:  $z_k \leftarrow \mathcal{M}_{\mathcal{B}_k^i}(w_k)$ , where  $\mathcal{M}_{\mathcal{B}_k^i}(w_k)$  is a masking keeping  $b_i$  elements from the parameter vector  $w_k$
  - 19: Construct the set  $\nabla_i = (\{G_i(z_k)\}_j, j \in \mathcal{B}_k^i)$ , where  $G_i(z_k)$  is as defined in (1)
  - 20: Send back  $\nabla_i$  to the server
- 

we present our convergence analysis and discuss on the interplay between the variables of our algorithm and its convergence properties. For space limitations, we only give a proof sketch for each result. **Full proofs are available**



in the supplementary material of this document.

### 5.1. Overview of the main result

To formally appreciate the interplay between all the variables of our problem, we first define the following variables to lighten the statement of our theorems, and we also give the main result of our formal analysis in the following general theorem.

$$\begin{aligned} U &= \max_{w \in \mathbb{R}^d} \left\{ \frac{1}{|C|} \sum_{j=1}^d \sum_{i \in \mathcal{C}} \left( M_w(Q'_{w_k} + \frac{1}{\lambda'_i}) + M'_{w_k} + \delta_{i,j}^2 + \epsilon_j^2 \right) \right\} \\ V &= \max_{w \in \mathbb{R}^d} \left\{ \frac{1}{|C|} \sqrt{\sum_{j=1}^d \left( \sum_{i \in \mathcal{C}} Q_w(Q'_{w_k} + \frac{1}{\lambda'_i}) \right)^2} \right\} \\ \beta &= \max \left\{ \tau \frac{\hat{\mu}_\lambda}{\hat{\mu}_s} \left( 1 - \sqrt{(\delta_\lambda^2 + \delta_s^2) \frac{v-1}{q-v+1}} \right), b_{min} \right\} \end{aligned}$$

with  $\hat{\mu}_x = \frac{1}{n} \sum_{i \in [n]} x_i$ ,  $\hat{\sigma}_x^2 = \frac{1}{n} \sum_{i \in [n]} (x_i - \mu_x)^2$ ,  $\delta_x = \frac{\sigma_x}{\mu_x}$ , for  $x \in [\lambda, s]$  and  $\lambda'_i = \frac{(|S_i|-1)}{\frac{|S_i|^d}{\tau \lambda_i} - 1}$

**Theorem.** *Let  $\lambda_i$  be the computation rate of worker  $i$ , and assume an  $(\epsilon, M', Q')$ -controlled masking scheme and a  $\Delta(v, f)$ -robust aggregation rule. Suppose that Assumptions 1, 3 and 4 hold. If the learning rate  $\gamma < \frac{1}{\sqrt{dL}}$  and  $\Delta(v, f)V < 1$ , then, when  $K \rightarrow \infty$ , Algorithm 1 eventually outputs a parameter estimate  $w$  such that:*

$$\mathbb{E} \|\nabla \mathcal{L}(w)\|^2 \leq \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)}$$

In Theorem 1 and 2, we present the formal convergence result of our algorithm. In doing so, we divide the parameter space into two regions  $\mathcal{W}$  and  $\mathbb{R}^d \setminus \mathcal{W}$ , with

$$\mathcal{W} = \left\{ w \mid \min_{\mathcal{B} \subseteq [d]} \left\{ (1 - \Delta V_w) \|\nabla \mathcal{L}(w)\|_{\mathcal{B}}^2 - \Delta U_w \right\} > 0 \right\}.$$

We denote by  $[\nabla \mathcal{L}(w)]_{\mathcal{B}}$  the partial gradient restricted to the coordinates in  $\mathcal{B}$ . Specifically, for all  $u = (\{u\}_1, \dots, \{u\}_d) \in \mathbb{R}^d$ ,  $u' = [u]_{\mathcal{B}} \in \mathbb{R}^d$  is such that  $\{u'\}_j = \{u\}_j$  if  $j \in \mathcal{B}$ , and 0 otherwise.

### 5.2. Convergence analysis

We start our theoretical analysis by deriving an upper bound in Lemma 2 on the error of the aggregated vector  $\text{AG}_{w_k}$  regarding the true gradient  $\nabla \mathcal{L}(w_k)$  computed at any parameter vector  $w_k \in \mathbb{R}^d$ .

**Lemma 2.** *Suppose that Assumptions 3 and 4 hold true. Consider the  $k$ -th step of Algorithm 1, and  $\mathcal{B}_k$ , the block computed by the server at step  $k$ . If AR is  $\Delta(v, f)$  robust, then*

$$\mathbb{E}_k [\|\text{AG}_{z_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2] \leq \Delta(v, f) (U_{w_k} + V_{w_k} \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2)$$

*Proof.* (Sketch) We first begin by upper bounding the error of a single worker's reduced gradient. We have then:  $\|[G_i(z_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \leq 2\|[G_i(z_k)]_{\mathcal{B}_k} - [G_i(w_k)]_{\mathcal{B}_k}\|^2 + 2\|[G_i(w_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2$ . Next, we compute upper bounds of the two right hand side terms using the classical bias-variance decomposition and assumptions 3 and 4, and the fact that workers use  $(\epsilon, M', Q')$ -masking schemes. After a few calculations, we obtain:  $\mathbb{E}_{\zeta\mathcal{M}} \|[G_i(z_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \leq \sigma_{ij}^2$ . Downsizing the previous inequality to each coordinate  $j \in \mathcal{B}_k$  and averaging this error through all workers leads to the average error of correct workers' partial gradients, which is exactly the second right hand side term in Definition 2 of a  $\Delta$ -robust aggregation rule. Using this definition, we obtain the desired result.  $\square$

Note that it is hard to demonstrate progress towards a local optimum without first demonstrating that  $\text{AG}_{w_k}$  is pointing in the same direction as the actual gradient of the loss function. Specifically, the scalar product between the aggregated gradient  $\text{AG}_{w_k}$  and  $\nabla \mathcal{L}(w_k)$  must be strictly positive. We show below in Lemma 3 that this condition holds true in our setting whenever  $w_k \in \mathcal{W}$ .

**Lemma 3.** *Suppose that Assumptions 3 and 4 hold true. Consider the  $k$ -th step of Algorithm 1. If AR is  $\Delta(v, f)$ -robust and  $w_k \in \mathcal{W}$ , then, denoting  $\Delta = \Delta(v, f)$ , we have:*

$$\mathbb{E}[\langle \text{AG}_{w_k}, [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle] > \frac{(1 - \Delta V_{w_k}) \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}^2 - \Delta U_{w_k}}{2}$$

*Proof.* (Sketch) Using Jensen's inequality, we have:  $\|\mathbb{E}_k[\text{AG}_z] - [\nabla \mathcal{L}(w)]_{\mathcal{B}}\|^2 \leq \mathbb{E}_k[\|\text{AG}_z - [\nabla \mathcal{L}(w)]_{\mathcal{B}}\|^2]$ . Using the upper bound from Lemma 2 which we call  $r^2$ , we also have:  $\|\mathbb{E}_k[\text{AG}_z] - [\nabla \mathcal{L}(w)]_{\mathcal{B}}\|^2 \leq r^2$ . This means that  $\mathbb{E}_k[\text{AG}_z]$  belongs to a ball centered at  $[\nabla \mathcal{L}(w)]_{\mathcal{B}}$  with radius  $r$ . From the perfect square identity, we can also see that :

$2 \langle \mathbb{E}_k[\text{AG}_{z_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle \geq \|\mathbb{E}_k[\text{AG}_{z_k}]\|^2 + \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}^2 - r^2$ . Using basic trigonometry, we have  $\sin \theta = \frac{r}{\|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}}$ , and since  $w_k \in \mathcal{W}$ , we also have  $\sin \theta \leq 1$ . Furthermore, from the triangle inequality,  $\|\mathbb{E}_k[\text{AG}_{z_k}]\| \geq \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k} - r$ . Combining these inequalities gives:

$2 \langle \mathbb{E}_k[\text{AG}_{z_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle \geq 2(1 - \sin \theta) \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}^2$ . Further development of this inequality and substituting  $r$  with the upper bound in Lemma 2 concludes the proof.  $\square$

Using this result, we obtain the convergence properties of ARGO for

strongly convex and non-convex functions. Under Assumption 1, we define

$$L = \max_{i \in [d]} L_i.$$

**Theorem 1** (Strongly convex convergence). *Suppose that Assumptions 1, 2, 3 and 4 hold true. Consider Algorithm 1 with a constant learning rate  $\gamma$ . If  $\gamma < \frac{1}{\sqrt{d}L}$  and  $\Delta V < 1$ , then*

$$\mathbb{E}[\mathcal{L}(w_K) - \mathcal{L}^*] \leq \left(1 - \frac{\beta}{d}\mu\gamma(1 - \Delta V)\right)^{K-1} (\mathcal{L}(w_1) - \mathcal{L}^* - H) + H$$

where  $\mathcal{L}^*$  denotes the minimum value of  $\mathcal{L}$  on  $\mathbb{R}^d$  and  $H = \frac{d\Delta U}{2\beta\mu(1-\Delta V)}$ .

*Proof.* (Sketch) First, we suppose that  $\forall k \in [T], w_k \in \mathcal{W}$ . In Algorithm 1, the parameter vector is updated as follows:  $w_{k+1} = w_k - \gamma \text{AG}_{z_k}$ . Now, consider an arbitrary  $k \in [T - 1]$ . Under Assumption 1, i.e., component-wise Lipschitz continuity of  $\nabla \mathcal{L}(w)$  with coefficient  $L$ , we have [40, Section 2.1]:  $\mathcal{L}(w_{k+1}) = \mathcal{L}(w_k - \gamma \text{AG}_{z_k}) \leq \mathcal{L}(w_k) - \gamma \langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, \text{AG}_{z_k} \rangle + \gamma^2 \frac{L_{\mathcal{B}_k}}{2} \|\text{AG}_{z_k}\|^2$ . Recall that  $L_{\mathcal{B}_k} \leq \sqrt{|\mathcal{B}_k|}L \leq \sqrt{d}L$ . Then, using the perfect square identity and Lemma 3, and the fact that  $\gamma < \frac{1}{\sqrt{d}L}$ , we obtain By taking the conditional expectation  $\mathbb{E}_k[\cdot]$ :  $\mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\gamma}{2}(1 - \Delta(v, f)V_{w_k}) \|[ \nabla \mathcal{L}(w_k) ]_{\mathcal{B}_k}\|^2 + \frac{\gamma \Delta(v, f)U_{w_k}}{2}$ . Now, introducing  $\mathbb{E}_{\mathcal{B}_k}$ , the conditional expectation given  $\mathcal{P}_k \setminus \mathcal{B}_k$  and the Polyak-Lojasiewicz inequality in the last inequality gives:  $\mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\mathbb{E}_{\mathcal{B}_k}[|\mathcal{B}_k|]}{d} \mu \gamma (1 - \Delta(v, f)V_{w_k})(\mathcal{L}(w_k) - \mathcal{L}^*) + \frac{\gamma \Delta(v, f)U_{w_k}}{2}$ . In Algorithm 1, if  $b_1, \dots, b_q$  are the sizes of the gradients reported by the  $q$  workers that have been contacted by the server in step 4 of the algorithm, then the server selects the  $v^{\text{th}}$  biggest size, which is the  $q - v + 1$  order statistic in an increasingly sorted set of items. We now make use of Lemma 3 to compute a lower bound on  $\mathbb{E}_{\mathcal{B}_k}[|\mathcal{B}_k|]$ . Then, knowing that  $\Delta(v, f)V < 1$  and  $\mu < \min_{k \in [T]} L_{\mathcal{B}_k} \leq \sqrt{d}L$ , we rearrange the

terms to obtain a contraction inequality. By applying it repeatedly through iterations  $k \in [T - 1]$ , we obtain the first part of the result. Now, if there exists  $s \in [T]$  such that  $w_s \notin \mathcal{W}$ , then, by definition of subset  $\mathcal{W}$ , the norm of the gradient is upper bounded by a term involving  $U, \Delta(v, f)$  and  $V$ . Rearranging the terms gives the second part of the result. Finally, taking the maximum of the upper bounds in the two parts concludes the proof.  $\square$

Unlike convex functions, non-convex functions may have multiple local minima, making convergence analysis more complex [38], especially for coordinate descent methods [41, 42], which are closer to our algorithm. While it is hard in general to upper bound the optimality gap, as in previous theorems, we upper bound the average (and thus the minimum) gradient norm of the cost function, evaluated using the sequence of parameter vectors  $\{w_k\}_{k \in [K]}$  generated through the iterations  $1, \dots, K$ .

**Theorem 2** (Non-convex convergence). *Suppose that Assumptions 1, 3 and 4 hold. If  $\gamma < \frac{1}{\sqrt{dL}}$  and  $\Delta V < 1$  then*

$$\min_{k \in [K]} \mathbb{E} \left[ \|\nabla \mathcal{L}(w_k)\|^2 \right] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{K\beta\gamma(1 - \Delta V)} + \frac{d\Delta U}{\beta(1 - \Delta(v, f)V)}$$

*Proof.* (Sketch) Same as Theorem 1. Instead of using the Polyak-Lojasiewicz inequality (only valid for the strongly convex case), we rearrange terms to upper bound the average (squared) norm of the gradients  $\mathbb{E} \left[ \frac{1}{T} \sum_{k=1}^T \|\nabla \mathcal{L}(w_k)\|^2 \right]$ . Using the fact that  $\min_{k \in [T]} X_k \leq \frac{1}{T} \sum_{k=1}^T X_k$ , we conclude the proof.  $\square$

### 5.3. Interplay between convergence properties and other factors

Our formal analysis highlights an interplay between the convergence properties of our algorithm and standard factors such as learning rate, mini-batch size and data heterogeneity, which is well studied in the literature [38, 43, 44, 45], but also new factors such as hardware capabilities, Byzantine impact and server’s waiting time. Particularly, we show for the first time that Byzantine resilience also depends on the hardware capabilities of workers.

**Convergence properties vs runtime complexity and Byzantine resilience.** Note that the variable  $\beta$  plays a crucial role in both theorems 1 and 2. In fact, increasing  $\beta$  improves the convergence rate and the statistical error of the algorithm. To do so, one can either increase the server’s waiting time  $\tau$  or the number of workers selected per iteration  $q$ . However, both actions increase the runtime complexity. One can also decrease the minimum number to execute the aggregation  $v$ , but this action increases the value of the robustness coefficient  $\Delta(v, f)$ , which has a negative impact on the convergence properties. Note that  $v$  is lower bounded anyway depending on the aggregation rule (e.g. Median needs  $v > 2f$ , where  $f$  is the number of Byzantine workers).

**Hardware capabilities vs Byzantine resilience.** The robustness coefficient  $\Delta(v, f)$  is a quantity that depends on the number of Byzantine workers in an execution, the attacks they implement and also the performance of the aggregation rule combined with our algorithm. Recent works [10, 12, 37, 46, 13, 11, 47, 15] guarantee the convergence of their algorithms in the presence of Byzantine workers, only using conditions on the number

of Byzantine workers and on the variance-norm ratio of the loss function's gradient. It turns out that hardware capabilities are an important part of the equation. In fact, to cope with a given Byzantine impact (encapsulated in the variable  $\Delta$ , there are minimum hardware requirements on the correct workers to ensure convergence. We formally state this result in Corollary 1.

**Corollary 1.** *Let  $\lambda_i$  be the computation rate of worker  $i \in \mathcal{C}$ , where  $\mathcal{C}$  denotes the set of correct workers. Let  $\tau > 0$  be the server's waiting time per iteration and  $\Delta = \Delta(v, f)$  be the robustness coefficient of a robust aggregation rule used in Algorithm 1. To ensure convergence, the average computation rate of correct workers  $\bar{\lambda}$  must satisfy the following:*

$$\bar{\lambda} \geq \frac{d}{\tau} \left( \frac{1}{\sqrt{dQ_M}\Delta} - Q'_M \right)^{-1} = \Omega \left( \frac{\Delta d}{\tau} \right)$$

where  $\bar{\lambda} = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \lambda_i$ ,  $Q_M = \max_{w \in \mathbb{R}^d} Q_w$  and  $Q'_M = \max_{w \in \mathbb{R}^d} Q'_w$ .

*Proof.* (Sketch) In both theorems 1 and 2, we introduced the necessary condition  $\Delta V > 1$ . Using the fact that  $\frac{1}{\lambda'_i} = \frac{|S_i|^d - 1}{\tau \lambda_i}$  is greater than  $\frac{d}{\tau \lambda_i}$ , we obtain a lower bound on the quantity  $\frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \frac{1}{\lambda_i}$ . Finally, by noting that the inverse of this quantity is exactly the harmonic mean of  $\lambda_1, \dots, \lambda_{n_c}$ , we conclude the proof using the fact that the arithmetic mean is greater than the harmonic mean.  $\square$

## 6. Empirical Evaluation

Through this section, we first demonstrate the possibility to exploit a network of weak devices (e.g. mobile devices) in distributed learning tasks

using ARGO. We also conduct experiments to showcase the interplay between factors such as convergence properties, hardware heterogeneity, data heterogeneity and Byzantine impact. Finally, we evaluate the energy consumption of executing a distributed learning task on mobile devices (using our algorithm) against executing it in a server computer (using standard SGD), and show that the first scheme is significantly more energy efficient.

### 6.1. Experimental setup

To simplify the presentation, we evaluate the performance of ARGO (used with a vanilla SGD update rule) against vanilla SGD to eliminate the impact of more advanced solvers such as NAG [48], RMSprop [49], AdaGrad [50] or Adam [51], that are still misunderstood from a technical viewpoint. For a fair comparison, we also use the same learning rate and mini-batch size for SGD and ARGO in every experiments.

**Models and datasets.** To confirm our theoretical results, we evaluated our procedure on a broad class of models, including linear regression, binary logistic regression and support vector machine, on several benchmark datasets (such as Boston<sup>3</sup>, Wisconsin Breast Cancer<sup>4</sup>, or Phishing<sup>5</sup>). We also considered multinomial logistic regression and feed-forward neural network, trained on MNIST<sup>6</sup> and Fashion-MNIST<sup>7</sup>. In the main paper, we only present a selection of experiments, but our findings were consistent across the full

---

<sup>3</sup><http://lib.stat.cmu.edu/datasets/boston>

<sup>4</sup>[https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+\(diagnostic\)](https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic))

<sup>5</sup><https://archive.ics.uci.edu/ml/datasets/phishing+websites>

<sup>6</sup><http://yann.lecun.com/exdb/mnist/>

<sup>7</sup><https://github.com/zalandoresearch/fashion-mnist>



set of experiments we considered.

**Network architecture and hardware specification.** We emulate the hardware capabilities of workers through the computation rate variable we introduced in Section 3. Basically, we classify workers into three categories characterizing their computational capabilities, namely: “weak”, “average” and “powerful”. Each category represents a set of workers whose computation rates are normally distributed over a fixed mean value that depends on the learning task. We consider several computation profiles for the network:

- $C_{standard}$ : 30% of weak, 40% of average and 30% of powerful devices;
- $C_{weak}$ : all devices are weak;
- $C_1$ : 90% of weak and 10% of average devices;
- $C_2$ : 70% of weak and 30% of average devices;
- $C_{powerful}$  all devices are powerful.

At each iteration, the server only contacts a random set of  $q$  workers, and aggregates their responses using a given aggregation rule, after a waiting time  $\tau$ . By default, we assume that we are in the standard scenario  $C_{standard}$ , set  $\frac{q}{n} = 0.8$ , and use *Average* and *Median* (a special case of Trimmed Mean) as aggregation rules in honest and Byzantine environments, respectively, unless specified otherwise.

**Data heterogeneity.** We experiment both homogeneous and heterogeneous partitioning of datasets over the workers. In the homogeneous case, each worker is assigned a non-overlapping independent and identically distributed sample from the training set. We construct heterogeneous versions

of the datasets by restricting the classes detained by each workers. Specifically, in our experiments, each worker only has access to data points for two classes out of ten from the MNIST and Fashion-MNIST datasets.

**Byzantine workers.** We use two state-of-the-art Byzantine attacks [52, 53] to evaluate the impact of our new learning scheme on the robustness of existing aggregation rules (Krum [10], Trimmed Mean [37] and Aksel [11]). These attacks exploit the normal distribution of correct estimates to construct Byzantine values that are still within a few standard deviations from the true mean. Byzantine workers start implementing the aforementioned attacks at iteration 10 of each experiment. For comparison, we always plot the test accuracy of averaging under no attack.

## 6.2. Performance analysis of ARGO

We first present an experiment (Figure 1) supporting the main concept of the very title of this work: overcoming hardware dependence in distributed learning. The principal advantage of ARGO is to allow the server to receive information from *all* the workers at each iteration, while SGD discards weaker workers at the expense of losing valuable information from their local datasets. This main difference explains the performance of our algorithm against the standard SGD, as illustrated in Figure 1a. While SGD’s performance is strongly correlated with the computation capabilities of the workers, ARGO shows roughly the same convergence properties, independently of the hardware profile of the workers. Interestingly, using ARGO allows to reach top performance, even with a network full of weak devices, which is impossible with standard learning schemes.

Next, we conduct another experiment (Figure 2) measuring this time the

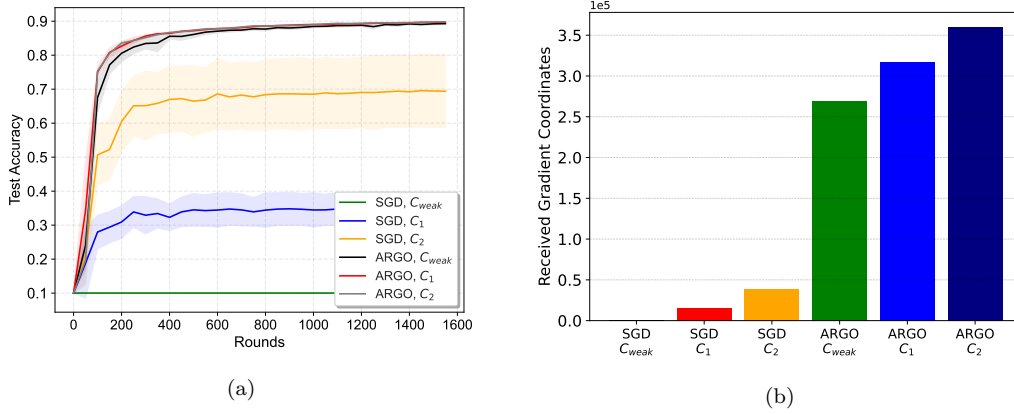


Figure 1: Training a multinomial logistic regression on a heterogeneously partitioned MNIST dataset, using  $n = 100$  workers in an honest setting. ARGO shows improved convergence properties (final accuracy, convergence rate) w.r.t. SGD, even in the weakest scenario  $C_{weak}$  where all workers have low hardware profiles (see Figure 1a). Figure 1b explains this improvement by the fact that ARGO receives more descent direction estimations than SGD, even in very restricted network profiles ( $C_{weak}$ : all the workers are weak devices), where no worker is able to compute an estimation in the case of SGD.

actual runtime (wall-clock) in milliseconds of both algorithms, and show the effect of waiting time  $\tau$  on both total runtime and final accuracy. When there is no waiting time limit ( $\tau = \infty$ ), SGD is slow, but achieves good accuracy by learning from all workers’ datasets. Setting a time limit per iteration ( $\tau = 32$ ) speeds up SGD, but results in lower accuracy when the dataset is not evenly partitioned, because of discarded data points from weak workers. With ARGO, even with a waiting time limit, workers can adjust their tasks, leading to better overall accuracy without slowing down the procedure. Note that this experiment favors ARGO even when hardware capabilities allow the use of SGD.

In the previous experiments, one can see that ARGO shows nearly the same convergence pattern for all scenarios. In the experiment reported in

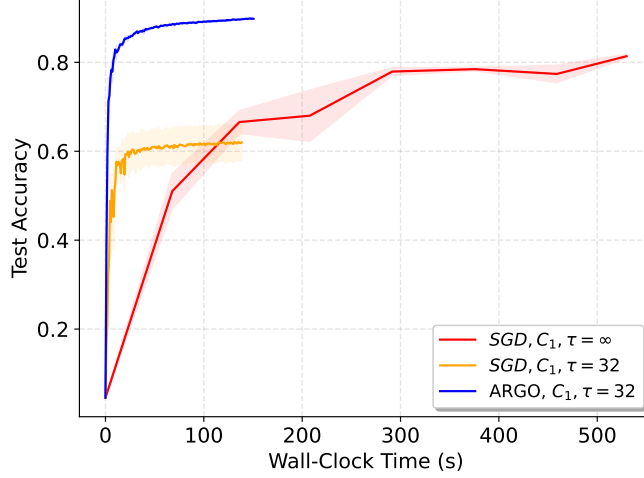


Figure 2: Training a multinomial logistic regression on MNIST, heterogeneously partitioned over  $n = 100$  honest workers. The server averages the estimates of  $q = 80$  random workers at each iteration. When  $\tau = \infty$ , the server waits for the slowest device to send its message. When  $\tau = 32$ , the server waits for 32 time units, and only aggregates the received messages.

Figure 3, we voluntarily set the computation rates of the weak devices to very low values, in order to get a lower accuracy using the weakest scenario  $C_{weak}$ . While using ARGO with a network full of weak workers still shows good performance, we demonstrate that replacing only 10% of these weak devices by average workers is sufficient to match the convergence quality of an all-powerful network on MNIST. This experiment is very interesting, and motivates the use of ARGO in industry-scale machine learning applications. For instance, a company may train a complex machine learning model using ARGO with data available on smartphones, which can be considered as weak devices. Then, it can introduce a few powerful computers to boost the learning performance. The result would be similar to a scenario where all the data is relocated into one location, and trained with strong computers

using SGD.

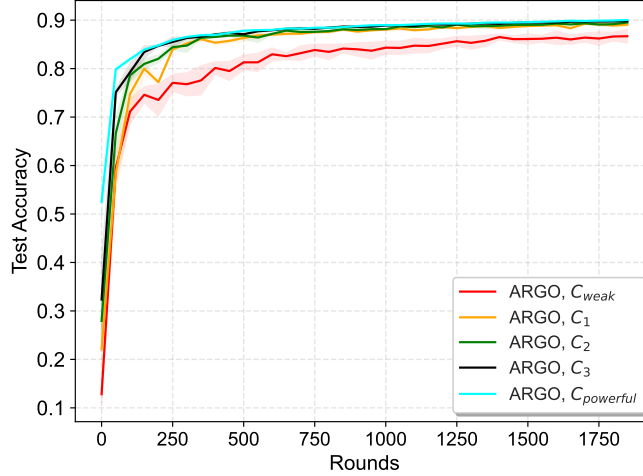


Figure 3: Training a multinomial logistic regression on MNIST, homogeneously partitioned over  $n = 100$  honest workers. The server averages the estimates of  $q = 80$  random workers at each iteration. Interestingly, replacing 10% of weak devices by average ones ( $C_1$ ) is sufficient to match the performance of ( $C_{powerful}$ ).

Finally, we demonstrated in Corollary 1 that the average hardware capabilities of correct workers is lower-bounded by a term involving the waiting time  $\tau$ , the model dimension  $d$  and the Byzantine impact  $\Delta$ . The experiment in Figure 4 illustrates this concept: although coordinate-wise median [13] is a robust statistic that can defend against attacks in an environment where nearly half of the workers are Byzantine ( $f = 39$  out of  $q = 80$  workers), the algorithm fails to converge even for  $f = 20$  for the hardware scenario  $C_1$ . In other words, the computation rates in  $C_1$  do not satisfy the condition of Corollary 1 for  $f = 20$ .

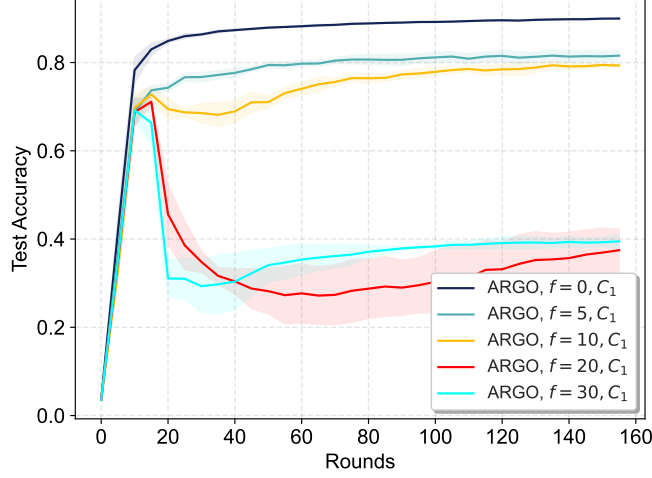


Figure 4: Training a multinomial logistic regression on MNIST, with a total of  $n = 100$  workers. The server aggregates the partial gradients of  $q = 80$  random workers using Median, at each iteration. We consider scenario  $C_1$ , with 90% of weak devices and 10% of powerful ones. At iteration 10, a number  $f$  of Byzantine workers attack the system by implementing the attack described in [52]. We vary the number of Byzantine workers and show that when  $f > 20$ , ARGO fails to converge.

### 6.3. Energy efficiency of ARGO on mobile devices

In the above section, we evaluated the performance (i.e. accuracy, runtime, Byzantine resilience) of ARGO, executed in a distributed environment using a computer. While ARGO is clearly a better variant than SGD for computer implementations, its true advantage is the possibility to move the computations to weak devices, where SGD cannot be executed due to its expensive iteration, hardware-wise. In that sense, we developed a cross-platform mobile application (working on Android and iOS operating systems) that implements the “worker device” part of ARGO.

**Methodology for evaluating energy consumption.** To measure the energy consumption, we follow the methodology described in [54], which

we summarize as follows: for computers, a processor core is shielded and configured to a fixed frequency, then the algorithm is pinned to this core in order to evaluate its exact energy consumption using the *PowerStat* software tool, excluding the consumption of operating system routines and other processes. For mobile devices, we use *Power Manager*<sup>8</sup> [54], a power profiling mobile application that collects key performance indicators (e.g., current, voltage, execution time) at specific intervals. Similarly to computers, mobile devices should be configured to the lowest running state, by shutting down unnecessary services (e.g., Bluetooth, location, screen brightness, etc.) to accurately measure the energy consumption of the algorithm.

**Setup and hardware characteristics.** In the following experiment, we train a feed-forward neural network on the MNIST dataset for 2000 iterations, using one worker device. We evaluate the energy consumption following the aforementioned methodology for (1): the worker is a computer (powerful device) running SGD and (2): the worker is a mobile device executing ARGO. Then, we infer the total consumption of a distributed learning by multiplying the number of devices in a run by the estimated energy consumption of the corresponding device category. We present in Table ?? the hardware characteristics of the devices used in our experiment.

In Figure 5, we can see that using mobile devices for a distributed learning task reduces the energy consumption by up to 94%, when compared to a computer-based distributed learning, while maintaining similar convergence properties. In scenario  $C_1$ , involving 90% of mobile devices and 10% of computers (which is a realistic scenario for an industrial AI project), the

---

<sup>8</sup><https://github.com/karimboubouh/PowerManager>

Platform	OS	CPU	Frequency	RAM
<b>Linux Server</b>	Ubuntu 20.04 LTS	Intel Xeon W-2123	Min 1.2 GHz Max 3.6 GHz	32GB
<b>Android Device</b>	Android 13	Google Tensor G1	Min 1.8 GHz Max 2.8 GHz	6GB

Table 1: Hardware and software characteristics of the considered server and Android device used in our experiments.

energy consumption of the learning task is still significantly low, but slightly improves convergence properties to match the computer-based learning performance.

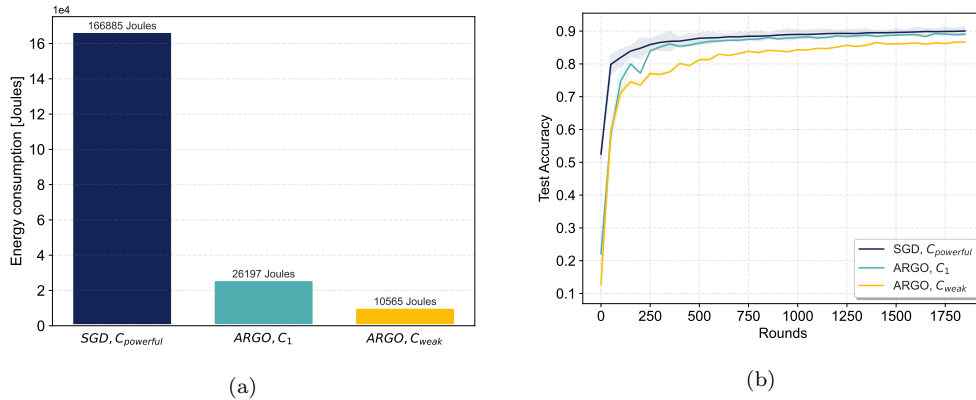


Figure 5: Training a feed-forward neural network on the MNIST dataset for 1800 iterations, using  $q = 80$  workers. The energy consumption of this distributed learning task using ARGO with exclusively mobile devices is significantly lower than the same task executed using SGD on computers. Scenario  $C_1$  offers a better compromise in terms of energy efficiency and convergence properties, which is only possible using our algorithm.



## 7. Conclusion

We presented ARGO, a new optimization algorithm that allows the execution of distributed learning tasks on weak devices such as smartphones. This is achieved by (1) adapting the iteration cost to the hardware capabilities of each device, (2) allowing dynamic participation of devices and (3) defending against possible Byzantine devices (honest failures or malicious attacks). We proved ARGO’s convergence under a non-trivial combination of hypotheses, namely: Byzantine failures, computational heterogeneity and data heterogeneity. Our theory demonstrated important tensions between relevant factors such as statistical error rate, impact of Byzantine workers, the time complexity of the algorithm and its convergence rate, which we also supported empirically. Particularly, we proved for the first time a lower bound on the hardware capabilities of correct workers to defend against Byzantine influence, in order to ensure convergence. Finally, we analyzed the energy consumption of ARGO running on mobile devices, and showed that it is significantly more energy efficient than traditional learning schemes running on computers, while maintaining similar convergence properties.

On a more ethical side, our work encourages data privacy by making smartphones (which are the most important personal data carrier nowadays) contribute directly to AI projects, without exposing local data to third parties. Furthermore, this work opens an interesting line of open questions, among which: (1) How can we design masking schemes for transformers, which constitute the backbone of modern large language models? (2) Can our work be extended to second order optimization algorithms? And finally, (3) what is the impact of network bandwidth on the performance of our

algorithm? We leave these questions open for future works.

## References

- [1] M. Soltanolkotabi, A. Javanmard, J. Lee, Theoretical insights into the optimization landscape of over-parameterized shallow neural networks, *IEEE Transactions on Information Theory* 65 (2017) 742–769.
- [2] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. Burges, L. Bottou, K. Weinberger (Eds.), *Advances in Neural Information Processing Systems*, Vol. 25, Curran Associates, Inc., 2012.  
URL [https://proceedings.neurips.cc/paper\\_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf)
- [3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848.
- [4] G. Csurka, J.-M. Renders, G. Jacquet, Xrce’s participation at patent image classification and image-based patent retrieval tasks of the clef-ip 2011, in: *Conference and Labs of the Evaluation Forum*, 2011.
- [5] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, Y. Wu, Coca: Contrastive captioners are image-text foundation models (2022). [arXiv:2205.01917](https://arxiv.org/abs/2205.01917).
- [6] OpenAI, Gpt-4 technical report (2023). [arXiv:2303.08774](https://arxiv.org/abs/2303.08774).

- [7] J. Konečný, H. McMahan, D. Ramage, P. Richtárik, Federated optimization: Distributed machine learning for on-device intelligence, ArXiv abs/1610.02527 (2016).
- [8] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, W. J. Dally, Eie: Efficient inference engine on compressed deep neural network, in: Proceedings of the 43rd International Symposium on Computer Architecture, ISCA '16, IEEE Press, 2016, p. 243–254. doi:10.1109/ISCA.2016.30.  
URL <https://doi.org/10.1109/ISCA.2016.30>
- [9] L. Lamport, R. Shostak, M. Pease, The byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401. doi:10.1145/357172.357176.  
URL <https://doi.org/10.1145/357172.357176>
- [10] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, J. Stainer, Machine learning with adversaries: Byzantine tolerant gradient descent, NeurIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 118–128.
- [11] A. Boussetta, E.-M. El-Mhamdi, R. Guerraoui, A. Maurer, S. Rouault, AKSEL: Fast Byzantine SGD, in: Q. Bramer, R. Oshman, P. Romano (Eds.), 24th International Conference on Principles of Distributed Systems (OPODIS 2020), Vol. 184 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021, pp. 8:1–8:16. doi:10.4230/LIPIcs.OPODIS.

2020.8.

URL <https://drops.dagstuhl.de/opus/volltexte/2021/13493>

- [12] G. Damaskinos, E. M. El Mhamdi, R. Guerraoui, R. Patra, M. Taziki, Asynchronous Byzantine machine learning (the case of SGD), in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, Vol. 80 of Proceedings of Machine Learning Research, PMLR, Stockholmsmässan, Stockholm Sweden, 2018, pp. 1145–1154.  
URL <http://proceedings.mlr.press/v80/damaskinos18a.html>
- [13] D. Yin, Y. Chen, K. Ramchandran, P. Bartlett, Byzantine-robust distributed learning: Towards optimal statistical rates (2018). [arXiv:1803.01498](#).
- [14] L. Chen, H. Wang, Z. Charles, D. Papailiopoulos, Draco: Byzantine-resilient distributed training via redundant gradients (2018). [arXiv:1803.09877](#).
- [15] D. Alistarh, Z. Allen-Zhu, J. Li, Byzantine stochastic gradient descent, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 31, Curran Associates, Inc., 2018, pp. 4613–4623.  
URL <https://proceedings.neurips.cc/paper/2018/file/a07c2f3b3b907aaf8436a26c6d77f0a2-Paper.pdf>
- [16] D. Data, S. Diggavi, Byzantine-tolerant distributed coordinate descent, in: 2019 IEEE International Symposium on Information Theory (ISIT), 2019, pp. 2724–2728. doi:10.1109/ISIT.2019.8849217.

- [17] Z. Yang, W. U. Bajwa, Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning, *IEEE Transactions on Signal and Information Processing over Networks* 5 (4) (2019) 611–627. doi:10.1109/TSIPN.2019.2928176.
- [18] X. Lian, Y. Huang, Y. Li, J. Liu, Asynchronous parallel stochastic gradient for non convex optimization., In *NIPS*, pages 2737–2745 (2015).
- [19] J. Jiang, B. Cui, C. Zhang, L. Yu, Heterogeneity-aware distributed parameter servers, in: *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17*, Association for Computing Machinery, New York, NY, USA, 2017, p. 463–478. doi:10.1145/3035918.3035933.  
URL <https://doi.org/10.1145/3035918.3035933>
- [20] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, A. Anandkumar, signsgd: Compressed optimisation for non-convex problems (2018). arXiv:1802.04434.
- [21] T. Vogels, S. P. Karimireddy, M. Jaggi, Powersgd: Practical low-rank gradient compression for distributed optimization, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, Vol. 32, Curran Associates, Inc., 2019.  
URL <https://proceedings.neurips.cc/paper/2019/file/d9fbed9da256e344c1fa46bb46c34c5f-Paper.pdf>

- [22] H. Sun, Y. Shao, J. Jiang, B. Cui, K. Lei, Y. Xu, J. Wang, Sparse gradient compression for distributed sgd, in: G. Li, J. Yang, J. Gama, J. Natwichai, Y. Tong (Eds.), Database Systems for Advanced Applications, Springer International Publishing, Cham, 2019, pp. 139–155.
- [23] Y. Lin, S. Han, H. Mao, Y. Wang, B. Dally, Deep gradient compression: Reducing the communication bandwidth for distributed training, in: International Conference on Learning Representations, 2018.  
URL <https://openreview.net/forum?id=SkhQHMWOW>
- [24] P. Baldi, P. J. Sadowski, Understanding dropout, in: C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 26, Curran Associates, Inc., 2013.  
URL [https://proceedings.neurips.cc/paper\\_files/paper/2013/file/71f6278d140af599e06ad9bf1ba03cb0-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2013/file/71f6278d140af599e06ad9bf1ba03cb0-Paper.pdf)
- [25] B. Y. A. K. C. M. Jermaine, Distributed learning of neural networks using independent subnet training, 2020.
- [26] A. Khaled, P. Richtárik, Gradient descent with compressed iterates, ArXiv abs/1909.04716 (2019).
- [27] N. A. Lynch, Distributed algorithms, Elsevier, 1996.
- [28] D. Masters, C. Luschi, Revisiting small batch training for deep neural networks (2018). [arXiv:1804.07612](https://arxiv.org/abs/1804.07612).
- [29] E. Díaz-Francés, F. Rubio, On the existence of a normal approximation to the distribution of the ratio of two independent normal

- random variables, *Statistical Papers* 54 (05 2013). doi:10.1007/s00362-012-0429-2.
- [30] B. Arnold, N. Balakrishnan, *Relations, Bounds and Approximations for Order Statistics*, *Lecture Notes in Statistics*, Springer New York, 2012.  
URL <https://books.google.co.ma/books?id=Tb7kBwAAQBAJ>
- [31] Q. Tao, K. Kong, D. Chu, G. Wu, Stochastic coordinate descent methods for regularized smooth and nonsmooth losses, in: P. A. Flach, T. De Bie, N. Cristianini (Eds.), *Machine Learning and Knowledge Discovery in Databases*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 537–552.
- [32] J. Konečný, Z. Qu, P. Richtárik, Semi-stochastic coordinate descent, *Optimization Methods and Software* 32 (5) (2017) 993–1005.  
arXiv:<https://doi.org/10.1080/10556788.2017.1298596>, doi:10.1080/10556788.2017.1298596.  
URL <https://doi.org/10.1080/10556788.2017.1298596>
- [33] P. Richtárik, M. Takáč, Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function, *Math. Program.* 144 (1–2) (2014) 1–38. doi:10.1007/s10107-012-0614-z.  
URL <https://doi.org/10.1007/s10107-012-0614-z>
- [34] R. Tappenden, M. Takáč, P. Richtárik, On the complexity of parallel coordinate descent, *Optimization Methods and Software* 33 (2) (2018) 372–395.

- [35] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic differentiation in machine learning: A survey, *J. Mach. Learn. Res.* 18 (1) (2017) 5595–5637.
- [36] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, *Nature* 323 (1986) 533–536.
- [37] C. Xie, O. Koyejo, I. Gupta, Phocas: dimensional byzantine-resilient stochastic gradient descent (2018). [arXiv:1805.09682](https://arxiv.org/abs/1805.09682).
- [38] L. Bottou, F. E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, *SIAM Review* 60 (2) (2018) 223–311. doi:10.1137/16M1080173.
- [39] L. Nguyen, P. H. Nguyen, M. van Dijk, P. Richtarik, K. Scheinberg, M. Takac, SGD and hogwild! Convergence without the bounded gradients assumption, Vol. 80 of *Proceedings of Machine Learning Research*, PMLR, Stockholmsmässan, Stockholm Sweden, 2018, pp. 3750–3758. URL <http://proceedings.mlr.press/v80/nguyen18c.html>
- [40] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, 1st Edition, Springer Publishing Company, Incorporated, 2014.
- [41] S. J. Wright, Coordinate descent algorithms, *Mathematical Programming* 151 (1) (2015) 3–34.
- [42] Y. Nesterov, Efficiency of coordinate descent methods on huge-scale optimization problems, *SIAM Journal on Optimization* 22 (2) (2012) 341–362.



- [43] A. Khaled, K. Mishchenko, P. Richtarik, Tighter theory for local sgd on identical and heterogeneous data, in: S. Chiappa, R. Calandra (Eds.), Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics, Vol. 108 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 4519–4529.  
URL <https://proceedings.mlr.press/v108/bayoumi20a.html>
- [44] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, S. U. Stich, A unified theory of decentralized sgd with changing topology and local updates, in: ICML, 2020, pp. 5381–5393.  
URL <http://proceedings.mlr.press/v119/koloskova20a.html>
- [45] R. M. Gower, N. Loizou, X. Qian, A. Sailanbayev, E. Shulgin, P. Richtárik, SGD: General analysis and improved rates, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, Vol. 97 of Proceedings of Machine Learning Research, PMLR, 2019, pp. 5200–5209.  
URL <https://proceedings.mlr.press/v97/qian19b.html>
- [46] C. Xie, O. Koyejo, I. Gupta, Generalized byzantine-tolerant sgd (2018).  
[arXiv:1802.10116](https://arxiv.org/abs/1802.10116).
- [47] E. M. El Mhamdi, R. Guerraoui, S. Rouault, The hidden vulnerability of distributed learning in Byzantium, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 3521–3530.  
URL <http://proceedings.mlr.press/v80/mhamdi18a.html>

- [48] Y. Nesterov, A method for solving the convex programming problem with convergence rate  $o(1/k^2)$ , Proceedings of the USSR Academy of Sciences 269 (1983) 543–547.
- [49] G. Hinton, N. Srivastava, K. Swersky, Neural networks for machine learning lecture 6a overview of mini-batch gradient descent, Cited on 14 (8) (2012) 2.
- [50] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (61) (2011) 2121–2159.  
URL <http://jmlr.org/papers/v12/duchi11a.html>
- [51] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980 (2015).
- [52] C. Xie, S. Koyejo, I. Gupta, Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation (2019). [arXiv:1903.03936](https://arxiv.org/abs/1903.03936).
- [53] G. Baruch, M. Baruch, Y. Goldberg, A little is enough: Circumventing defenses for distributed learning, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 32, Curran Associates, Inc., 2019, pp. 8635–8645.  
URL <https://proceedings.neurips.cc/paper/2019/file/ec1c59141046cd1866bbbcdfb6ae31d4-Paper.pdf>
- [54] R. Basmadjian, K. Boubouh, A. Boussetta, R. Guerraoui, A. Maurer, On the advantages of p2p ml on mobile devices, in: Proceedings of the

Thirteenth ACM International Conference on Future Energy Systems,  
2022, pp. 338–353.

# Supplementary material of the paper: Overcoming Hardware Dependence in Distributed Learning

Karim Boubouh<sup>a</sup>, Amine Boussetta<sup>a,\*</sup>, Rachid Guerraoui<sup>b</sup>, Alexandre Maurer<sup>a</sup>

<sup>a</sup>*College of Computing, UM6P, Benguerir, Morocco*

<sup>b</sup>*Distributed Computing Laboratory, EPFL, Lausanne, Switzerland*

## 1. Summary of contributions and changes regarding the original work

In our original paper, we proposed a genuine adaptive algorithm (HgO) inspired by block coordinate descent methods, where devices are allowed to compute a few partial derivatives instead of the full gradient, depending on their hardware capabilities. Combined with a robust aggregation rule, the algorithm is shown to be Byzantine-resilient with an adaptive iteration cost, and to converge under a set of mild assumptions. Although strong results were derived in this work, the main algorithm, as well as its theoretical guarantees, only apply to separable cost functions, in order to compute partial derivatives independently. This includes many “simple” problems in ML such as linear regression, logistic regression, support vector machines and so forth. We extend this work to non-separable cost functions, which represent the largest category of ML models used in the industry nowadays: feed-forward neural networks, convolutional neural networks and transformers, which are the building block of modern AI-enabled applications (Apple’s Siri, Open AI’s ChatGPT, Ultralytics’s Yolo ...). We detail the new contributions as follows:

**Design.** In the original algorithm, each worker computes a few partial derivatives independently, whose indices are randomly shuffled by the server. This approach cannot work in the case of non-separable cost functions. We modify the algorithm such that each worker applies a masking scheme that reduces the model size, depending on its hardware, before computing the full gradient. In this case, random shuffling of coordinate indices is replaced by a consistent randomization, taking into consideration the ML model architecture.

**Formal analysis.** The original theoretical guarantees did not cover some aspects of the algorithm, namely: the sub-selection of  $q$  available workers among the full set of workers, the minimum number  $d$  to execute an aggregation rule, as well as the expected number of coordinates selected after applying steps 7 to 11 of the algorithm, which was denoted as  $\mathbb{E}[|\mathcal{B}_k|]$  in the original proof. In this extension, we derive a lower bound of this quantity in Lemma 1, which allows us to include the aforementioned parameters in our theoretical analysis, thus offering a better understanding of the dynamics of the algorithm. We also introduce a new definition on masking schemes, and modify the proofs of Lemma 2 and Lemma 3 to take into consideration the impact of model sub-selection (step 5 of honest worker procedure in our algorithm). Finally, we formalize the discussion on the interplay between hardware capabilities and Byzantine impact in Corollary 1 and give, for the first time, a lower bound on the average hardware capabilities of correct workers  $\bar{\lambda} = \Omega\left(\frac{d\Delta}{\tau}\right)$ , given a Byzantine impact  $\Delta$ , a model dimension  $d$  and a server’s waiting time  $\tau$ , to ensure convergence.

**Energy efficiency.** We execute the mobile version of our algorithm on smartphones, and show that running distributed learning tasks on mobile devices consumes significantly less energy than executing the same task on computers, while maintaining similar convergence properties. This further highlights the practicality of using our algorithm in industry-scale learning for privacy, performance and, most importantly, energy efficiency.

## 2. Additional results and explanations

In this section, we provide more details supporting our theoretical analysis as well as proposition on the  $\Delta$ -robustness of Trimmed Mean.

### 2.1. On the learning rate

In the above theorems, we used a tight (safe) upper bound on the learning rate  $\gamma < \frac{1}{\sqrt{dL}}$  to eliminate the dependence on the random block  $\mathcal{B}$  constructed by the server at step 10 in Algorithm 1. However, HGO only requires the following

\*Corresponding author

Email addresses: karim.boubouh@um6p.ma (Karim Boubouh), amine.boussetta@um6p.ma (Amine Boussetta), rachid.guerraoui@epfl.ch (Rachid Guerraoui), alexandre.maurer@um6p.ma (Alexandre Maurer)

condition  $\gamma_k < \frac{1}{\sqrt{|\mathcal{B}_k|L}}$  at each iteration. Note that the new upper bound is looser and allows for big steps, depending on the block size  $|\mathcal{B}_k|$ , which varies at each iteration. In practice, one can use a dynamic learning rate  $\gamma_k$ , inversely proportional to  $\sqrt{|\mathcal{B}_k|}$ , to improve the convergence rate.

## 2.2. On the generality of our results

Our work encapsulates a large family of Byzantine-resilient and non-Byzantine-resilient optimization algorithms. For instance, in [1], the author studies Random Coordinate Descent (RCD), where a single random index ( $b = 1$ ) is selected at each iteration, and the adequate coordinate is updated following a gradient step. In this simple algorithm, the gradient step is computed using a learning rate and a partial derivative, which is itself computed using the complete dataset. Since an exact non-distributed computation is executed, there will be no variance ( $M_k = Q_k = 0$ ), no aggregation ( $\Delta(v, f) = 0$ ), no mini-batch ( $\forall i \in [n], s_i = 1$ ) and no bias ( $\forall (i, j) \in [N] \times [d], \delta_{i,j} = 0$ ), which means that  $U_w = V_w = 0$  and  $H_k = 0$ . Replacing these values in Theorem 1 recreates the convergence result for RCD, presented in the second part of Theorem 1 in [1].

## 2.3. On the $\Delta$ -robustness of Trimmed Mean

Trimmed Mean (TM) is a classical robust statistic that has been studied in the literature, especially in the field of Byzantine machine learning. With a truncation parameter  $t$ , the function  $\text{TM} : \mathbb{R}^\rho \rightarrow \mathbb{R}$  averages all but the  $t$  smallest and largest values among the  $\rho$  initial values where we assume that  $f < t < \rho/2$ . In [2], the authors derived, assuming *unbiased estimation*, an upper bound on the trimmed mean of  $\rho$  scalars with  $f$  among them being possibly Byzantine, by leveraging results on order statistics. We extend their result to *biased estimation*. Recall that  $\mathcal{C}$  denotes the set of correct workers, and for each  $i \in \mathcal{C}$  its gradient  $G_i(w)$  at parameter  $w \in \mathbb{R}^d$  is defined by (1). We also denote the gradients sent by a Byzantine worker  $i$  by  $\tilde{G}_i(w)$ , which however may be arbitrary.

**Proposition 1.** *Let  $\rho > 2f$ , then for all  $w \in \mathbb{R}^d$  and  $j \in [d]$  and for any  $\mathcal{Q} \subset [n]$ , such that  $|\mathcal{Q}| > \rho$  we have*

$$\mathbb{E} \left[ (\text{TM}_j - \{\nabla \mathcal{L}(w)\}_j)^2 \right] \leq \Delta(\rho, f) \left( \frac{1}{|\mathcal{C} \cap \mathcal{Q}|} \sum_{i \in \mathcal{C} \cap \mathcal{Q}} \mathbb{E} [|\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j|^2] \right)$$

where  $\Delta(\rho, f) = \frac{2|\mathcal{C} \cap \mathcal{Q}|(t + |\mathcal{C} \cap \mathcal{Q}|)}{(|\mathcal{C} \cap \mathcal{Q}| - t)^2}$  and  $\text{TM}_j$  denotes  $\text{TM} : (\{G_i(w)\}_j; i \in \mathcal{Q})$ .

## 2.4. Linking biased estimation with data heterogeneity

Besides the differences in workers' computational capabilities, distributed learning also faces the challenge of data heterogeneity, i.e., the data partitioning amongst the workers need not be uniform. Thus, even the correct workers are unable to compute unbiased estimates of the true gradient  $\nabla \mathcal{L}(w)$ , as pictorially shown in Figure 1 below, where the expectation of the  $j^{\text{th}}$  partial derivative computed using the  $i^{\text{th}}$  data point is a simple translation of the target parameter.

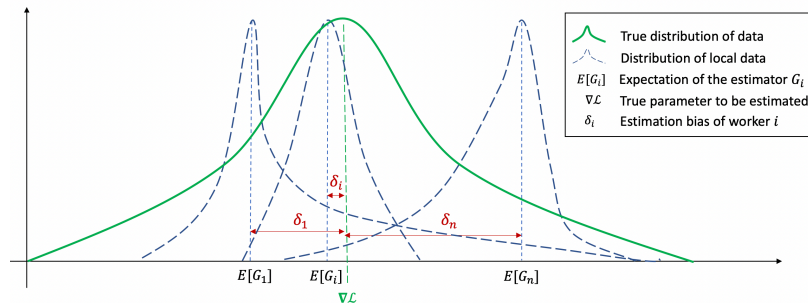


Figure 1: The dashed blue curves model the distributions of local data as seen by each worker. The green curve models the true distribution of data. In our setting, each worker tries to estimate a block of partial derivatives of the true cost function using only its local data. Due to data heterogeneity, the estimates of the workers are variously biased.

## 3. Proof of theoretical results

We recall all the variables used in this work and their corresponding designation.

Table 1: List of variables and their designation

$N$	Total number of data points across the network
$n$	Number of workers
$\lambda_i$	Computation rate of worker $i$
$f$	Maximum number of Byzantine workers
$\mathcal{Q}_j$	Set of workers that computed coordinate $j$
$\mathcal{C}_j$	Set of correct workers that computed coordinate $j$
$q$	Number of random workers selected by the server
$\tau$	Number of time units the server waits for worker's estimations
AR	Robust aggregation rule
$\Delta$	Robustness coefficient of AR
$v$	Minimum number of workers to execute the aggregation
$\mu$	Strong convexity parameter
$L_i$	Coordinate-wise lipschitz parameter
$w_k$	Parameter vector at iteration $k$
$\mathcal{M}_i$	Masking scheme used by worker $i$
$z_k$	sub model of worker $i$ : $z_k = \mathcal{M}_i(w_k)$
$d$	Dimension of the parameter vector
$\mathcal{S}_i$	Local data set of worker $i$
$\zeta_i$	Random mini-batch selected by worker $i$
$s_i$	$=  \zeta_i $ mini-batch size
$\mathcal{B}_i$	Block of coordinates computed by worker $i$
$b_i$	$=  \mathcal{B}_i $ Block size
$\mathcal{L}$	Cost function being optimized
$l(w, x)$	loss computed at $w$ using the data point $x$
$G_i$	Gradient estimation of worker $i$
$\nabla \mathcal{L}(w)$	True gradient
$\{X\}_j$	$j^{th}$ coordinate of vector $X$
$[X]_{\mathcal{B}}$	$\{[X]_{\mathcal{B}}\}_j = \{X\}_j$ if $j \in \mathcal{B}$ , $= 0$ otherwise
$\gamma$	Learning rate (step size)
$\delta_{i,j}$	Bias parameter of worker $i$ at coordinate $j$
$M_k, Q_k$	Variance parameters of $[\nabla l(w, x)]_j, \forall j \in [d]$ and $w \in \mathbb{R}^d$
$\epsilon_k, M'_k, Q'_k$	Bias and variance parameters of gradients using an $(\epsilon_k, M'_k, Q'_k)$ - masking scheme
$F_0$	$= \mathcal{L}(w_1) - \mathcal{L}(w_*)$ (initial gap)

### 3.1. Proof of Proposition 1

*Proof.* Let us consider  $w \in \mathbb{R}^d$ ,  $j \in [d]$  and  $\mathcal{Q} \subset [n]$ , such that  $|\mathcal{Q}| > \rho$ . From a prior result [2, proof of Theorem 1], we have

$$\begin{aligned}
& \left( \text{TM}_j - \{\nabla \mathcal{L}(w)\}_j \right)^2 \\
& \leq \frac{2}{(|\mathcal{C} \cap \mathcal{Q}| - t)^2} \left( \left( \sum_{i \in \mathcal{C} \cap \mathcal{Q}} (\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j) \right)^2 + t \sum_{i \in \mathcal{C} \cap \mathcal{Q}} (\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j)^2 \right). \quad (1)
\end{aligned}$$

Furthermore, from Jensen's inequality and the fact that correct workers sample mini-batches independently, we get

$$\mathbb{E} \left[ \left( \sum_{i \in \mathcal{C} \cap \mathcal{Q}} (\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j) \right)^2 \right] \leq |\mathcal{C} \cap \mathcal{Q}| \sum_{i \in \mathcal{C} \cap \mathcal{Q}} \mathbb{E} \left[ |\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j|^2 \right]. \quad (2)$$

Hence taking the expectation on both sides in (1) and using (2) we get the expected result.  $\square$

### 3.2. Proof of Lemma 1

**Lemma.** Let  $\lambda_i$  and  $s_i$  be the computation rate and the mini-batch size of worker  $i$  for  $i \in [n]$ , and  $\tau > 0$  be the server's waiting time. Suppose that  $\lambda \sim \mathcal{N}(\mu_\lambda, \sigma_\lambda^2)$  and  $s \sim \mathcal{N}(\mu_s, \sigma_s^2)$  with coefficients of variation satisfying  $\delta_\lambda = \frac{\sigma_\lambda}{\mu_\lambda} < 1$  and

$\delta_s = \frac{\sigma_s}{\mu_s} < 1$ , and  $\frac{\lambda}{s} \in \left[ \frac{\mu_\lambda}{\mu_s} \pm \left( \frac{\mu_\lambda}{\mu_s} \right) \sqrt{\delta_\lambda^2 + \delta_s^2} \right]$ . Then the  $k^{th}$  order statistic of  $n$  independently drawn samples from the random variable  $b = \frac{\lambda\tau}{s}$  is approximated by a normal distribution, and we have:

$$\max \left\{ \tau \frac{\hat{\mu}_\lambda}{\hat{\mu}_s} \left( 1 - \sqrt{(\delta_\lambda^2 + \delta_s^2) \frac{n-k}{k}} \right), b_{min} \right\} \leq \mathbb{E}[b_{(k)}]$$

where  $\hat{\mu}_\lambda = \frac{1}{n} \sum_{i \in [n]} \lambda_i$ ,  $\hat{\mu}_s = \frac{1}{n} \sum_{i \in [n]} s_i$ ,  $\hat{\sigma}_\lambda^2 = \frac{1}{n} \sum_{i \in [n]} (\lambda_i - \mu_\lambda)^2$ ,  $\hat{\sigma}_s^2 = \frac{1}{n} \sum_{i \in [n]} (s_i - \mu_s)^2$  and  $\delta_x = \frac{\sigma_x}{\mu_x}$  for  $x \in [\lambda, s]$ .

*Proof.* Suppose that  $\lambda$  and  $s$  are two independent random variables following the normal distributions  $\mathcal{N}(\mu_\lambda, \sigma_\lambda^2)$  and  $\mathcal{N}(\mu_s, \sigma_s^2)$  respectively. The maximum likelihood estimators of the location and scale parameters are the sample mean and the sample variance. We have then:

$$\begin{aligned} \hat{\mu}_\lambda &= \frac{1}{n} \sum_{i \in [n]} \lambda_i, & \hat{\mu}_s &= \frac{1}{n} \sum_{i \in [n]} s_i \\ \hat{\sigma}_\lambda^2 &= \frac{1}{n} \sum_{i \in [n]} (\lambda_i - \mu_\lambda)^2, & \hat{\sigma}_s^2 &= \frac{1}{n} \sum_{i \in [n]} (s_i - \mu_s)^2 \end{aligned}$$

Note that the random variable  $b = \frac{\lambda\tau}{s}$  is a fraction of two random variables  $\lambda$  and  $s$ , multiplied by a scalar  $\tau$ , and therefore follows a ratio distribution. To find closed form expressions for its expectation and its variance, we use a normal approximation of the ratio distribution derived in [3]: since  $\hat{\mu}_\lambda, \hat{\mu}_s, \hat{\sigma}_\lambda^2$  and  $\hat{\sigma}_s^2$  are positive distribution parameters, and the coefficient of variation  $\delta_\lambda = \frac{\hat{\sigma}_\lambda}{\hat{\mu}_\lambda} < 1$  and  $\delta_s = \frac{\hat{\sigma}_s}{\hat{\mu}_s} < 1$ , we have the following for all  $\frac{\lambda}{s} \in \left[ \frac{\hat{\mu}_\lambda}{\hat{\mu}_s} \pm \left( \frac{\hat{\mu}_\lambda}{\hat{\mu}_s} \right) \sqrt{\delta_\lambda^2 + \delta_s^2} \right]$ :

$$\mathbb{E} \left[ \frac{\lambda}{s} \right] = \frac{\hat{\mu}_\lambda}{\hat{\mu}_s}, \quad \text{var} \left( \frac{\lambda}{s} \right) = \left( \frac{\hat{\mu}_\lambda}{\hat{\mu}_s} \right)^2 (\delta_\lambda^2 + \delta_s^2)$$

Since  $b = \frac{\lambda\tau}{s}$ , we also have:

$$\mathbb{E}[b] = \mu_b = \tau \frac{\hat{\mu}_\lambda}{\hat{\mu}_s}, \quad \text{var}(b) = \sigma_b^2 = \tau^2 \left( \frac{\hat{\mu}_\lambda}{\hat{\mu}_s} \right)^2 (\delta_\lambda^2 + \delta_s^2)$$

Now, the random variable  $b$  approximately follows a normal distribution  $\mathcal{N}(\mu_b, \sigma_b^2)$ . We are interested in the expected value of the  $k^{th}$  element of the increasingly sorted  $n$  samples  $b_1, \dots, b_n$ . Using the lower bound on the expected order statistics from Theorem 3.19 in [4], we obtain:

$$\mu_b - \sigma_b \sqrt{\frac{n-k}{k}} \leq \mathbb{E}[b_{(k)}] \quad (3)$$

Since we know that that  $b \geq b_{min}$  from Equation 3, we obtain a tighter lower bound for 3:

$$\max \left\{ \mu_b - \sigma_b \sqrt{\frac{n-k}{k}}, b_{min} \right\} \leq \mathbb{E}[b_{(k)}]$$

which concludes the proof. □

### 3.3. Proof of Lemma 2

We first define the expectations used in what follows:

- $\mathbb{E}_k[\cdot]$ : the conditional expectation given the history  $\mathcal{P}_k = \{w_1, \dots, w_k; \text{AG}_{w_1}, \dots, \text{AG}_{w_{k-1}}; \mathcal{B}_k\}$
- $\mathbb{E}_{\mathcal{B}_k}$ : the conditional expectation given  $\mathcal{P}_k \setminus \mathcal{B}_k$
- $\mathbb{E}[\cdot] = \mathbb{E}_{\mathcal{B}_1} \mathbb{E}_1[\dots \mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k \dots]$ : the expectation over the total randomness of the algorithm

Now we restate the lemma for convenience.

**Lemma.** Suppose that Assumptions 3 and 4 hold true. Consider the  $k$ -th step of Algorithm 1 and  $\mathcal{B}_k$  the block computed by the server at step  $k$ . If AR is  $\Delta(v, f)$  robust at  $w_k$  then

$$\mathbb{E}_k \left[ \|\mathbf{AG}_{z_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \right] \leq \Delta(v, f) \left( U_{w_k} + V_{w_k} \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}^2 \right) \quad (4)$$

where  $s'_i := \frac{s_i(|\mathcal{S}_i|-1)}{|\mathcal{S}_i|-s_i}$ ,  $U_{w_k} = \frac{1}{|\mathcal{C}|} \sum_{j \in [d]} \left( \sum_{i \in \mathcal{C}} \left( \frac{M_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) + M'_{w_k} + \delta_{i,j}^2 + \epsilon_j^2 \right) \right)$ ,  
 $V_{w_k} = \frac{1}{|\mathcal{C}|} \left( \sum_{j \in [d]} \left( \sum_{i \in \mathcal{C}} \frac{Q_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) \right)^2 \right)^{\frac{1}{2}}$ , and  $\mathcal{C}$  denotes the set of honest workers.

*Proof.* First, we consider an arbitrary correct worker  $i$  and block  $\mathcal{B}_k \subseteq [d]$ . Recall that  $G_i(w_k) = \frac{1}{s_i} \sum_{x \in \zeta_i} \nabla \ell(w_k, x)$  and  $G_i(z_{ik}) = G_i(\mathcal{M}_i(w_k))$  where  $\zeta_i$  is a mini-batch of  $s_i$  data points sampled randomly from  $\mathcal{S}_i$ , and  $z_{ik}$  is the reduced parameter vector obtained by worker  $i$  after applying a masking function  $\mathcal{M}_i$ . Note that

$$\| [G_i(z_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \|^2 = \| [G_i(z_k)]_{\mathcal{B}_k} - [G_i(w_k)]_{\mathcal{B}_k} + [G_i(w_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \|^2 \quad (5)$$

$$\leq 2 \| [G_i(z_k)]_{\mathcal{B}_k} - [G_i(w_k)]_{\mathcal{B}_k} \|^2 + 2 \| [G_i(w_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \|^2 \quad (6)$$

At this stage, it is important to identify the source of randomness in inequalities 5. For a given block  $\mathcal{B}_k$  and parameter vector  $w_k$ , randomness comes from the mini-batch selection  $\zeta_i$  and the masking scheme used by worker  $i$ . The latter is directly linked to the shuffled set of weight indices  $\mathcal{C}_k$  constructed by the server (see step 3 of Algorithm 1). We define  $\mathbb{E}_{\zeta_i}$  and  $\mathbb{E}_{\mathcal{M}_i}$  as the expectation over the mini batch and the expectation over the masking scheme, respectively. We also set  $\mathbb{E}_{\zeta \mathcal{M}} = \mathbb{E}_{\zeta_i} \mathbb{E}_{\mathcal{M}_i}$  as the total conditional expectation given  $\mathcal{P}_k$ . We have then:

$$\mathbb{E}_{\zeta \mathcal{M}} \| [G_i(z_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \|^2 \leq 2 \mathbb{E}_{\zeta \mathcal{M}} \| [G_i(z_k)]_{\mathcal{B}_k} - [G_i(w_k)]_{\mathcal{B}_k} \|^2 + 2 \mathbb{E}_{\zeta \mathcal{M}} \| [G_i(w_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \|^2 \quad (7)$$

Using the decomposition of the total error into bias and variance on the two right hand side terms of the previous inequality, we obtain for the first term:

$$\mathbb{E}_{\zeta \mathcal{M}} \| [G_i(z_k)]_{\mathcal{B}_k} - [G_i(w_k)]_{\mathcal{B}_k} \|^2 = \mathbb{E}_{\zeta \mathcal{M}} \| [G_i(z_k)]_{\mathcal{B}_k} - \mathbb{E}_{\mathcal{M}_i} [[G_i(z_k)]_{\mathcal{B}_k}] + \mathbb{E}_{\mathcal{M}_i} [[G_i(z_k)]_{\mathcal{B}_k}] - [G_i(w_k)]_{\mathcal{B}_k} \|^2 \quad (8)$$

$$= \mathbb{E}_{\zeta \mathcal{M}} \| [G_i(z_k)]_{\mathcal{B}_k} - \mathbb{E}_{\mathcal{M}_i} [[G_i(z_k)]_{\mathcal{B}_k}] \|^2 + \mathbb{E}_{\zeta \mathcal{M}} \| \mathbb{E}_{\mathcal{M}_i} [[G_i(z_k)]_{\mathcal{B}_k}] - [G_i(w_k)]_{\mathcal{B}_k} \|^2 \quad (9)$$

$$= \mathbb{E}_{\zeta_i} \left[ \mathbb{E}_{\mathcal{M}_i} \| [G_i(z_k)]_{\mathcal{B}_k} - \mathbb{E}_{\mathcal{M}_i} [[G_i(w_k)]_{\mathcal{B}_k}] \|^2 + \| \mathbb{E}_{\mathcal{M}_i} [[G_i(z_k)]_{\mathcal{B}_k}] - [G_i(w_k)]_{\mathcal{B}_k} \|^2 \right] \quad (10)$$

$$= \mathbb{E}_{\zeta_i} \left[ \sum_{j \in \mathcal{B}_k} \mathbb{E}_{\mathcal{M}_i} ([G_i(z_k)]_j - \mathbb{E}_{\mathcal{M}_i} [[G_i(w_k)]_j])^2 + \sum_{j \in \mathcal{B}_k} (\mathbb{E}_{\mathcal{M}_i} [[G_i(z_k)]_j] - [G_i(w_k)]_j)^2 \right] \quad (11)$$

Since  $\mathcal{M}_i$  is an  $(\epsilon, M', Q')$ -masking, we have:

$$\mathbb{E}_{\zeta \mathcal{M}} \| [G_i(z_k)]_{\mathcal{B}_k} - [G_i(w_k)]_{\mathcal{B}_k} \|^2 \leq \mathbb{E}_{\zeta_i} \left[ b_k M'_{w_k} + Q'_{w_k} \| [G_i(w_k)]_{\mathcal{B}_k} \|^2 + \sum_{j \in \mathcal{B}_k} \epsilon_j^2 \right] \quad (12)$$

$$= b_k M'_{w_k} + \sum_{j \in \mathcal{B}_k} \epsilon_j^2 + Q'_{w_k} \mathbb{E}_{\zeta_i} \| [G_i(w_k)]_{\mathcal{B}_k} \|^2 \quad (13)$$

Using the fact that  $\mathbb{E}[X^2] = \mathbb{E}[X - \mathbb{E}[X]]^2 + \mathbb{E}[X]^2 \leq \mathbb{E}[X - \mathbb{E}[X]]^2$ , we also have:

$$\mathbb{E}_{\zeta \mathcal{M}} \| [G_i(z_k)]_{\mathcal{B}_k} - [G_i(w_k)]_{\mathcal{B}_k} \|^2 \leq b_k M'_{w_k} + \sum_{j \in \mathcal{B}_k} \epsilon_j^2 + Q'_{w_k} \mathbb{E}_{\zeta_i} \| [G_i(w_k)]_{\mathcal{B}_k} - \mathbb{E}_{\zeta_i} [G_i(w_k)]_{\mathcal{B}_k} \|^2 \quad (14)$$



Using again Assumption 4, we obtain:

$$\mathbb{E}_{\zeta\mathcal{M}} \|[G_i(z_k)]_{\mathcal{B}_k} - [G_i(w_k)]_{\mathcal{B}_k}\|^2 \leq b_k(M_{w_k}Q'_{w_k} + M'_{w_k}) + \sum_{j \in \mathcal{B}_k} \epsilon_j^2 + Q_{w_k}Q'_{w_k} \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2 \quad (15)$$

Now for the second term in 7, we have:

$$\mathbb{E}_{\zeta\mathcal{M}} \|[G_i(w_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 = \mathbb{E}_{\zeta_i} \|[G_i(w_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \quad (16)$$

$$= \underbrace{\left[ \|[G_i(z_k)]_{\mathcal{B}_k} - \mathbb{E}_{\zeta_i} [[G_i(w_k)]_{\mathcal{B}_k}]\|^2 \right]}_{\mathbf{A}} + \underbrace{\left[ \|\mathbb{E}_{\zeta_i} [[G_i(w_k)]_{\mathcal{B}_k}] - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \right]}_{\mathbf{B}} \quad (17)$$

We now obtain below upper bounds on **A** and **B**.

**A**). Note that  $\zeta_i$  is a mini-batch obtained by sampling  $s_i$  random data points from  $\mathcal{S}_i$  without replacements, where  $s_i = |\zeta_i|$ . Furthermore, by definition of  $[\cdot]_{\mathcal{B}_k}$ ,  $\forall y \in \mathbb{R}^d$  we have  $\|[y]_{\mathcal{B}_k}\|^2 = \sum_{j \in \mathcal{B}_k} (\{y\}_j)^2$ . Hence, from [5], we have

$$\mathbb{E}_{\zeta_i} \left[ \|[G_i(w_k)]_{\mathcal{B}_k} - \mathbb{E}_{\zeta_i} [[G_i(w_k)]_{\mathcal{B}_k}]\|^2 \right] \leq \mathbb{E}_{z \sim \mathcal{S}_i} \left[ \sum_{j \in \mathcal{B}_k} \left( \{\nabla \ell(w_k, z)\}_j - \mathbb{E}_{x \sim \mathcal{S}_i} [\{\nabla \ell(w_k, x)\}_j] \right)^2 \right] \left( \frac{|\mathcal{S}_i| - s_i}{s_i(|\mathcal{S}_i| - 1)} \right).$$

Recall that for all  $s'_i := \frac{s_i(|\mathcal{S}_i| - 1)}{|\mathcal{S}_i| - s_i}$ . Finally, substituting from Assumption 4 we obtain that

$$\mathbb{E}_{\zeta_i} \left[ \|[G_i(w_k)]_{\mathcal{B}_k} - \mathbb{E}_{\zeta_i} [[G_i(w_k)]_{\mathcal{B}_k}]\|^2 \right] \leq \sum_{j \in \mathcal{B}_k} \left( \frac{M_{w_k}}{s'_i} + \frac{Q_{w_k}}{s'_i} (\{\nabla \mathcal{L}(w_k)\}_j)^2 \right) = \frac{b_k M_{w_k}}{s'_i} + \frac{Q_{w_k}}{s'_i} \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2 \quad (18)$$

where  $b_k$  is the size of block  $\mathcal{B}_k$ .

**B**). Similarly, we note that

$$\|\mathbb{E}_{\zeta_i} [[G_i(w_k)]_{\mathcal{B}_k}] - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 = \|\mathbb{E}_{x \sim \mathcal{S}_i} [\{\nabla \ell(w_k, x)\}_{\mathcal{B}_k}] - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2$$

Substituting from Assumption 3 above, we obtain that

$$\|\mathbb{E}_{\zeta_i} [[G_i(w_k)]_{\mathcal{B}_k}] - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \leq \sum_{j \in \mathcal{B}_k} \delta_{i,j}^2. \quad (19)$$

Substituting from (18) and (19) in (5) we obtain that

$$\mathbb{E}_{\zeta_i} \left[ \|[G_i(w_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \right] \leq \frac{b_k M_{w_k}}{s'_i} + \sum_{j \in \mathcal{B}_k} \delta_{i,j}^2 + \frac{Q_{w_k}}{s'_i} \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2. \quad (20)$$

Now plugging 15 and 20 into 6, we obtain:

$$\begin{aligned} \mathbb{E}_{\zeta\mathcal{M}} \|[G_i(z_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 &\leq b_k \left( M_{w_k} \left( Q'_{w_k} + \frac{1}{s'_i} \right) + M'_{w_k} \right) + \sum_{j \in \mathcal{B}_k} (\delta_{i,j}^2 + \epsilon_j^2) + \\ &\quad Q_{w_k} \left( Q'_{w_k} + \frac{1}{s'_i} \right) \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2 \end{aligned} \quad (21)$$

As  $\mathcal{B}_k$  is an arbitrary subset of  $[d]$ , (21) implies that for all  $j \in \mathcal{B}_k$ ,

$$\mathbb{E}_{\zeta\mathcal{M}} [(\{G_i(z_k)\}_j - \{\nabla \mathcal{L}(w_k)\}_j)^2] \leq M_{w_k} \left( Q'_{w_k} + \frac{1}{s'_i} \right) + M'_{w_k} + \delta_{i,j}^2 + \epsilon_j^2 + Q_{w_k} \left( Q'_{w_k} + \frac{1}{s'_i} \right) \{\nabla \mathcal{L}(w_k)\}_j^2 \quad (22)$$

$$= \frac{M_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) + M'_{w_k} + \delta_{i,j}^2 + \epsilon_j^2 + \frac{Q_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) \{\nabla \mathcal{L}(w_k)\}_j^2 \quad (23)$$

Next, recall that in the algorithm for each  $j \in \mathcal{B}_k$  the server receives the partial derivatives from at least  $v$  workers denoted by set  $\mathcal{Q}_j$ , i.e.,  $|\mathcal{Q}_j| \geq v$ . We denote the set of correct workers that send the  $j$ -th coordinate of their gradients by  $\mathcal{C}_j = \mathcal{C} \cap \mathcal{Q}_j$ . Since (23) holds for an arbitrary correct worker  $i \in \mathcal{C}$ , we obtain that:

$$\begin{aligned} \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \mathbb{E}_{\zeta, \mathcal{M}} [(\{G_i(z_k)\}_j - \{\nabla \mathcal{L}(w_k)\}_j)^2] &\leq \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \left( \frac{M_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) + M'_{w_k} + \delta_{i,j}^2 + \epsilon_j^2 \right) + \\ &\quad \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \frac{Q_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) \{\nabla \mathcal{L}(w_k)\}_j^2. \end{aligned} \quad (24)$$

Now, as AR is assumed  $\Delta(v, f)$ -robust at  $w_k$ , by Definition 2,

$$\mathbb{E}_k [(\{\text{AG}_{z_k}\}_j - \{\nabla \mathcal{L}(w_k)\}_j)^2] \leq \Delta(v, f) \left( \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \mathbb{E}_{\zeta, \mathcal{M}} [(\{G_i(z_k)\}_j - \{\nabla \mathcal{L}(w_k)\}_j)^2] \right).$$

Substituting from (24) above we obtain that

$$\begin{aligned} \mathbb{E}_k (\{\text{AG}_{z_k}\}_j - \{\nabla \mathcal{L}(w_k)\}_j)^2 &\leq \\ \Delta(v, f) &\left( \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \left( \frac{M_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) + M'_{w_k} + \delta_{i,j}^2 + \epsilon_j^2 \right) + \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \left( \frac{Q_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) \{\nabla \mathcal{L}(w_k)\}_j^2 \right) \right). \end{aligned} \quad (25)$$

By summing both sides in (25) over all  $j \in \mathcal{B}$  we obtain that:

$$\begin{aligned} \mathbb{E}_k [\|[\text{AG}_{z_k}]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2] &\leq \\ \Delta(v, f) &\left( \sum_{j \in \mathcal{B}_k} \left( \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \left( \frac{M_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) + M'_{w_k} + \delta_{i,j}^2 + \epsilon_j^2 \right) \right) + \sum_{j \in \mathcal{B}_k} \left( \left( \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \frac{Q_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) \right) \{\nabla \mathcal{L}(w_k)\}_j^2 \right) \right). \end{aligned} \quad (26)$$

Using Cauchy-Schwartz  $|\sum_i a_i b_i| \leq \left( \sum_i a_i^2 \right)^{\frac{1}{2}} \left( \sum_i b_i^2 \right)^{\frac{1}{2}}$ , we have

$$\sum_{j \in \mathcal{B}_k} \left( \left( \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \frac{Q_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) \right) \{\nabla \mathcal{L}(w_k)\}_j^2 \right) \leq \left( \sum_{j \in \mathcal{B}_k} \left( \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \frac{Q_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) \right)^2 \right)^{\frac{1}{2}} \left( \sum_{j \in \mathcal{B}_k} \{\nabla \mathcal{L}(w_k)\}_j^4 \right)^{\frac{1}{2}}. \quad (27)$$

Recall that  $\left( \sum_i a_i^2 \right)^{\frac{1}{2}} \leq \sum_i |a_i|$ . Thus,

$$\left( \sum_{j \in \mathcal{B}_k} \{\nabla \mathcal{L}(w_k)\}_j^4 \right)^{\frac{1}{2}} \leq \sum_{j \in \mathcal{B}_k} \{\nabla \mathcal{L}(w_k)\}_j^2 \quad (28)$$

Recall that  $[\text{AG}_{z_k}]_{\mathcal{B}_k} = \text{AG}_{z_k}$ . Hence, substituting from (27) and (28) in (26), we have

$$\begin{aligned} \mathbb{E}_k [\|\text{AG}_{z_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2] &\leq \Delta(v, f) \left( \sum_{j \in \mathcal{B}_k} \left( \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \left( \frac{M_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) + M'_{w_k} + \delta_{i,j}^2 + \epsilon_j^2 \right) \right) \right) + \\ &\quad \Delta(v, f) \left( \left( \sum_{j \in \mathcal{B}_k} \left( \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} \frac{Q_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) \right)^2 \right)^{\frac{1}{2}} \left( \sum_{j \in \mathcal{B}_k} \{\nabla \mathcal{L}(w_k)\}_j^2 \right) \right) \end{aligned}$$

$$\begin{aligned} \mathbb{E}_k[\|\mathbf{AG}_{z_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2] &\leq \Delta(v, f) \left( \frac{1}{|\mathcal{C}|} \sum_{j \in [d]} \left( \sum_{i \in \mathcal{C}} \left( \frac{M_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) + M'_{w_k} + \delta_{i,j}^2 + \epsilon_j^2 \right) \right) \right) + \\ &\quad \Delta(v, f) \left( \frac{1}{|\mathcal{C}|} \left( \sum_{j \in [d]} \left( \sum_{i \in \mathcal{C}} \frac{Q_{w_k}}{s'_i} (s'_i Q'_{w_k} + 1) \right)^2 \right)^{\frac{1}{2}} \left( \sum_{j \in [d]} \{\nabla \mathcal{L}(w_k)\}_j^2 \right) \right) \end{aligned}$$

which concludes the proof.  $\square$

### 3.4. Proof of Lemma 3

We are now ready to prove Lemma 3, which is stated again below for readability.

**Lemma.** *Suppose that Assumptions 3 and 4 hold true. Consider the  $k$ -th step of Algorithm 1 and  $\mathcal{B}_k$  the block computed by the server at step  $k$ . If AR is  $\Delta(v, f)$ -robust at  $w_k$  and  $w_k \in \mathcal{W}$  then*

$$\mathbb{E}_k[\langle \mathbf{AG}_{z_k}, [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle] > \frac{1}{2} \left( (1 - \Delta(v, f) V_{w_k}) \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}^2 - \Delta(v, f) U_{w_k} \right).$$

*Proof.* From Lemma 2 we have

$$\underbrace{\Delta(v, f) \left( U_w + V_w \|\nabla \mathcal{L}(w)\|_{\mathcal{B}}^2 \right)}_{:=r^2} \geq \mathbb{E}_k[\|\mathbf{AG}_z - [\nabla \mathcal{L}(w)]_{\mathcal{B}}\|^2] \geq \|\mathbb{E}_k[\mathbf{AG}_z] - [\nabla \mathcal{L}(w)]_{\mathcal{B}}\|^2 \quad (29)$$

where the second inequality follows from Jensen's inequality. The fact that  $\|\mathbb{E}_k[\mathbf{AG}_z] - [\nabla \mathcal{L}(w)]_{\mathcal{B}}\|^2 \leq r^2$  means that  $\mathbb{E}_k[\mathbf{AG}_z]$  belongs to a ball centered at  $[\nabla \mathcal{L}(w)]_{\mathcal{B}}$  with radius  $r$ , illustrated in Figure 2 below.

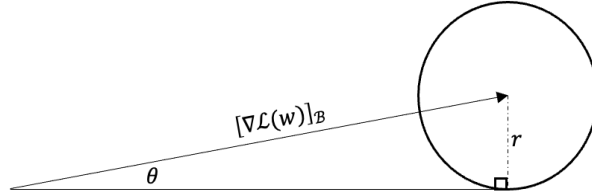


Figure 2: When  $\|\mathbb{E}_k[\mathbf{AG}_{z_k}] - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\| \leq r$ ,  $\mathbb{E}_k[\mathbf{AG}_{z_k}]$  belongs to a ball centered at  $[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}$  with radius  $r$ .

From (29) we obtain that

$$\|\mathbb{E}_k[\mathbf{AG}_{z_k}] - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 = \|\mathbb{E}_k[\mathbf{AG}_{z_k}]\|^2 - 2\langle \mathbb{E}_k[\mathbf{AG}_{z_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle + \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}^2 \leq r^2$$

Thus,

$$2\langle \mathbb{E}_k[\mathbf{AG}_{z_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle \geq \|\mathbb{E}_k[\mathbf{AG}_{z_k}]\|^2 + \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}^2 - r^2. \quad (30)$$

As we assume that  $w_k \in \mathcal{W}$ ,  $\Delta(v, f) U_{w_k} \leq (1 - \Delta(v, f) V_{w_k}) \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}^2$ . Thus,

$$\frac{r}{\|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}} \leq 1. \quad (31)$$

Owing to (31), we consider  $\theta \in [0, \pi/2]$  such that  $\sin \theta = \frac{r}{\|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}}$ . Furthermore, from the triangle inequality,  $\|\mathbb{E}_k[\mathbf{AG}_{z_k}]\| \geq \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k} - r$ . Therefore,  $\|\mathbb{E}_k[\mathbf{AG}_{z_k}]\| \geq (1 - \sin \theta) \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k} \geq 0$ . Using these in (30) we

obtain that

$$\begin{aligned}
2\langle \mathbb{E}_k[\mathbf{AG}_{z_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle &\geq \|\mathbb{E}_k[\mathbf{AG}_{z_k}]\|^2 + \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - r^2 \\
&\geq (1 - \sin \theta)^2 \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 + \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - \sin^2 \theta \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \\
&= 2(1 - \sin \theta) \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2.
\end{aligned}$$

Thus,

$$\langle \mathbb{E}_k[\mathbf{AG}_{z_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle \geq (1 - \sin \theta) \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2.$$

Substituting  $\sin \theta = \frac{r}{\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|}$  above we obtain that

$$\langle \mathbb{E}_k[\mathbf{AG}_{z_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle \geq \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - r \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\| \geq \frac{1}{2} \left( \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - r^2 \right).$$

Recall that  $r^2 := \Delta(v, f) \left( U_{w_k} + V_{w_k} \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \right)$ . Substituting this above concludes the proof, i.e., we have

$$\langle \mathbb{E}_k[\mathbf{AG}_{z_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle \geq \frac{1}{2} \left( (1 - V_{w_k} \Delta(v, f)) \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - \Delta(v, f) U_{w_k} \right).$$

□

### 3.5. Proof of Theorem 1

**Theorem.** Suppose that Assumptions 1, 2, 3 and 4 hold true. Consider Algorithm 1 with a constant learning rate  $\gamma$ . If  $\gamma < \frac{1}{dL}$  and  $\Delta(v, f)V < 1$  then

$$\mathbb{E}[\mathcal{L}(w_T) - \mathcal{L}^*] \leq \left( 1 - \frac{\beta}{d} \mu \gamma (1 - \Delta(v, f)V) \right)^{T-1} (\mathcal{L}(w_1) - \mathcal{L}^* - H) + H$$

where  $\mathcal{L}^*$  denotes the minimum value of  $\mathcal{L}$  on  $\mathbb{R}^d$ ,  $\beta = \max \left\{ \tau \frac{\mu_\lambda}{\mu_s} \left( 1 - \sqrt{(\delta_\lambda^2 + \delta_s^2) \frac{v-1}{q-v+1}} \right), b_{\min} \right\}$ ,  $U = \max_{k \in [T]} U_{w_k}$ ,  $V = \max_{k \in [T]} V_{w_k}$ , and  $H = \frac{d\Delta(v, f)U}{2\beta\mu(1-\Delta(v, f)V)}$ .

*Proof.* Recall the definition of set  $\mathcal{W}$ ,

$$\mathcal{W} := \left\{ w \in \mathbb{R}^d; \min_{\mathcal{B} \subseteq [d]} \left\{ (1 - \Delta(v, f)V_w) \|[\nabla \mathcal{L}(w)]_{\mathcal{B}}\|^2 - \Delta(v, f)U_w \right\} > 0 \right\}.$$

To prove the theorem, we separately consider below the two cases: **A**) when  $w_k \in \mathcal{W}$  for all  $k \in [T]$ , and **B**) there exists  $s \in [T]$  such that  $w_s \notin \mathcal{W}$ .

**A).** All the elements of the sequence  $\{w_k\}_{k \in [T]}$  are in the subspace  $\mathcal{W}$ . For all  $k \in [T-1]$ , we have  $w_{k+1} = w_k - \gamma \mathbf{AG}_{z_k}$ . Now, we consider an arbitrary  $k \in [T-1]$ . Under Assumption 1, i.e., component-wise Lipschitz continuity of  $\nabla \mathcal{L}(w)$  with coefficient  $L$ , and the fact that  $\mathbf{AG}_{z_k} = [\mathbf{AG}_{z_k}]_{\mathcal{B}_k}$  we have [6, Section 2.1]

$$\mathcal{L}(w_{k+1}) = \mathcal{L}(w_k - \gamma \mathbf{AG}_{z_k}) \leq \mathcal{L}(w_k) - \gamma \langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, \mathbf{AG}_{z_k} \rangle + \gamma^2 \frac{L_{\mathcal{B}_k}}{2} \|\mathbf{AG}_{z_k}\|^2 \quad (32)$$

where  $L_{\mathcal{B}_k}$  is the Lipschitz coefficient of the truncated gradient  $[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}$ .

Using the fact that  $\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} - \mathbf{AG}_{z_k}\|^2 = \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - 2\langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, \mathbf{AG}_{z_k} \rangle + \|\mathbf{AG}_{z_k}\|^2$ , (32) becomes

$$\begin{aligned}
\mathcal{L}(w_{k+1}) &\leq \mathcal{L}(w_k) - \gamma(1 - \gamma L_{\mathcal{B}_k}) \langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, \mathbf{AG}_{z_k} \rangle \\
&\quad + \gamma^2 \frac{L_{\mathcal{B}_k}}{2} \left( \|\mathbf{AG}_{z_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \right).
\end{aligned} \quad (33)$$

By taking the conditional expectation  $\mathbb{E}_k[\cdot]$  on both sides in (33) we obtain

$$\begin{aligned}
\mathbb{E}_k[\mathcal{L}(w_{k+1})] &\leq \mathcal{L}(w_k) - \gamma(1 - \gamma L_{\mathcal{B}_k}) \langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, \mathbb{E}_k[\mathbf{AG}_{z_k}] \rangle \\
&\quad + \gamma^2 \frac{L_{\mathcal{B}_k}}{2} \left( \mathbb{E}_k \left[ \|\mathbf{AG}_{z_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \right] - \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \right)
\end{aligned} \quad (34)$$

Recall that  $L_{\mathcal{B}_k} \leq \sqrt{|\mathcal{B}_k|}L \leq \sqrt{d}L$  where  $L = \max_{i \in [d]} L_i$ . Then, using Lemma 3 and the fact that  $\gamma < \frac{1}{\sqrt{d}L}$  we obtain:

$$\begin{aligned} \mathbb{E}_k[\mathcal{L}(w_{k+1})] &\leq \mathcal{L}(w_k) - \frac{1}{2}\gamma(1 - \gamma\sqrt{d}L)((1 - \Delta(v, f)V_{w_k}) \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2 - \Delta(v, f)U_{w_k}) \\ &\quad + \gamma^2 \frac{\sqrt{d}L}{2} \left( \mathbb{E}_k \left[ \|\text{AG}_{z_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \right] - \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2 \right) \end{aligned} \quad (35)$$

Using Lemma 2 in (35), we also have:

$$\begin{aligned} \mathbb{E}_k[\mathcal{L}(w_{k+1})] &\leq \mathcal{L}(w_k) - \frac{1}{2}\gamma(1 - \gamma\sqrt{d}L) \left( (1 - \Delta(v, f)V_{w_k}) \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2 - \Delta(v, f)U_{w_k} \right) \\ &\quad + \gamma^2 \frac{\sqrt{d}L}{2} \left( \Delta(v, f) \left( U_{w_k} + V_{w_k} \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2 \right) - \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2 \right) \end{aligned} \quad (36)$$

Rearranging the terms gives:

$$\mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\gamma}{2}(1 - \Delta(v, f)V_{w_k}) \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2 + \frac{\gamma\Delta(v, f)U_{w_k}}{2} \quad (37)$$

Now, introducing  $\mathbb{E}_{\mathcal{B}_k}$  the conditional expectation given  $\mathcal{P}_k \setminus \mathcal{B}_k$  we get

$$\mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\gamma}{2}(1 - \Delta(v, f)V_{w_k}) \mathbb{E}_{\mathcal{B}_k} \left[ \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}\|^2 \right] + \frac{\gamma\Delta(v, f)U_{w_k}}{2} \quad (38)$$

$$\leq \mathcal{L}(w_k) - \frac{\mathbb{E}_{\mathcal{B}_k} [|\mathcal{B}_k|]}{d} \frac{\gamma}{2}(1 - \Delta(v, f)V_{w_k}) \|\nabla \mathcal{L}(w_k)\|^2 + \frac{\gamma\Delta(v, f)U_{w_k}}{2} \quad (39)$$

Using the Polyak-Lojasiewicz inequality in (39) gives:

$$\mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\mathbb{E}_{\mathcal{B}_k} [|\mathcal{B}_k|]}{d} \mu\gamma(1 - \Delta(v, f)V_{w_k})(\mathcal{L}(w_k) - \mathcal{L}^*) + \frac{\gamma\Delta(v, f)U_{w_k}}{2} \quad (40)$$

Recall that in step 8,9,10 and 11 of Algorithm 1, the server computes the block size  $|\mathcal{B}|$  where each coordinate has been reported by at least  $v$  workers, to ensure the guarantees of the aggregation rule in the presence of Byzantine workers. If  $b_1, \dots, b_q$  are the sizes of the gradients reported by the  $q$  workers that have been contacted by the server in step 4 of the algorithm, then the server selects the  $v^{th}$  biggest size, which is the  $q - v + 1$  order statistic in an increasingly sorted set of items.

From Lemma 1, we know that for all  $\alpha \in [n]$ ,  $\mathbb{E}[b_{(\alpha)}] \geq \max \left\{ \tau \frac{\mu_\lambda}{\mu_s} \left( 1 - \sqrt{(\delta_\lambda^2 + \delta_s^2) \frac{n-\alpha}{\alpha}} \right), b_{min} \right\}$ .

In our case,  $n = q$  and  $\alpha = q - v + 1$ . We obtain by substituting above:

$$\mathbb{E}[|\mathcal{B}_k|] \geq \max \left\{ \tau \frac{\mu_\lambda}{\mu_s} \left( 1 - \sqrt{(\delta_\lambda^2 + \delta_s^2) \frac{v-1}{q-v+1}} \right), b_{min} \right\} = \beta$$

Substituting in Inequality 40 gives:

$$\mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\beta}{d} \mu\gamma(1 - \Delta(v, f)V_{w_k})(\mathcal{L}(w_k) - \mathcal{L}^*) + \frac{\gamma\Delta(v, f)U_{w_k}}{2} \quad (41)$$

Let  $U = \max_{k \in [T]} U_{w_k}$  and  $V = \max_{k \in [T]} V_{w_k}$ . Substituting from the above, we get

$$\mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\beta}{d} \mu\gamma(1 - \Delta(v, f)V)(\mathcal{L}(w_k) - \mathcal{L}^*) + \frac{\gamma\Delta(v, f)U}{2} \quad (42)$$

If we now subtract  $H = \frac{d\Delta(v, f)U}{2\beta\mu(1-\Delta(v, f)V)}$  and  $\mathcal{L}^*$  from both sides of (42), we obtain:

$$\mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k[\mathcal{L}(w_{k+1})] - \mathcal{L}^* - H \leq \left( 1 - \frac{\beta}{d} \mu\gamma(1 - \Delta(v, f)V) \right) (\mathcal{L}(w_k) - \mathcal{L}^* - H) \quad (43)$$

As  $\mathbb{E}[\cdot] = \mathbb{E}_{\mathcal{B}_1} \mathbb{E}_1[\dots \mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k \dots]$ , from above we obtain that

$$\mathbb{E}[\mathcal{L}(w_{k+1})] - \mathcal{L}^* - H \leq \left( 1 - \frac{\beta}{d} \mu\gamma(1 - \Delta(v, f)V) \right) (\mathbb{E}[\mathcal{L}(w_k)] - \mathcal{L}^* - H) \quad (44)$$

By assumption,  $\Delta(v, f)V < 1$ . We also know that  $\mu < \min_{k \in [T]} L_{\mathcal{B}_k} \leq \sqrt{d}L$ , which means that:  $0 < \frac{\beta}{d} \mu\gamma(1 -$

$\Delta(v, f)V < 1$ . Inequality (44) is therefore a contraction inequality. By applying it repeatedly through iterations  $k \in [T - 1]$ , we obtain:

$$\mathbb{E}[\mathcal{L}(w_T)] - \mathcal{L}^* \leq \left(1 - \frac{\beta}{d}\mu\gamma(1 - \Delta(v, f)V)\right)^{T-1} (\mathcal{L}(w_1) - \mathcal{L}^* - H) + H. \quad (45)$$

**B).** There exists at least one element of the sequence  $\{w_k\}_{k \in [T]}$  that belongs to the subspace  $\mathbb{R}^d \setminus \mathcal{W}$ . Let us denote by  $s$  the biggest integer such that  $w_s \in \mathbb{R}^d \setminus \mathcal{W}$ , then by definition of  $\mathcal{W}$ , for all  $\mathcal{B} \subset [d]$

$$\|[\nabla \mathcal{L}(w_s)]_{\mathcal{B}}\|^2 \leq \frac{\Delta(v, f)U_{w_s}}{1 - \Delta(v, f)V_{w_s}}. \quad (46)$$

Thus,

$$\mathbb{E}_{\mathcal{B}_s} \left[ \|[\nabla \mathcal{L}(w_s)]_{\mathcal{B}_s}\|^2 \right] \leq \frac{\Delta(v, f)U_{w_s}}{1 - \Delta(v, f)V_{w_s}} \quad (47)$$

$$\frac{\mathbb{E}_{\mathcal{B}_s}[b_s]}{d} \|\nabla \mathcal{L}(w_s)\|^2 \leq \frac{\Delta(v, f)U_{w_s}}{1 - \Delta(v, f)V_{w_s}} \quad (48)$$

$$\|\nabla \mathcal{L}(w_s)\|^2 \leq \frac{d\Delta(v, f)U_{w_s}}{\mathbb{E}_{\mathcal{B}_s}[|\mathcal{B}_s|](1 - \Delta(v, f)V_{w_s})} \leq \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)}. \quad (49)$$

Using the Polyak-Lojasiewicz inequality, we have

$$\|\nabla \mathcal{L}(w_s)\|^2 \geq 2\mu(\mathcal{L}(w_s) - \mathcal{L}^*). \quad (50)$$

Combining (49) and (50) gives

$$\mathcal{L}(w_s) - \mathcal{L}^* \leq \underbrace{\frac{d\Delta(v, f)U}{2\beta\mu(1 - \Delta(v, f)V)}}_H. \quad (51)$$

If  $s = T$ , then the above implies

$$\mathcal{L}(w_T) - \mathcal{L}^* \leq \underbrace{\frac{d\Delta(v, f)U}{2\beta\mu(1 - \Delta(v, f)V)}}_H. \quad (52)$$

Otherwise, recall that for all  $k \in [T]$  where  $k > s$  we have  $w_k \in \mathcal{W}$ . Thus by applying the same reasoning as in **A)** we get

$$\mathbb{E}[\mathcal{L}(w_T)] - \mathcal{L}^* \leq \left(1 - \frac{\beta}{d}\mu\gamma(1 - \Delta(v, f)V)\right)^{T-s} (\mathbb{E}[\mathcal{L}(w_s)] - \mathcal{L}^* - H) + H. \quad (53)$$

Finally by substituting (51) in (53) we get  $\mathbb{E}[\mathcal{L}(w_T)] - \mathcal{L}^* \leq H$ .

**A+B).**

From (45), (52) and (53) we obtain,  $\forall w \in \mathbb{R}^d$ :

$$\mathbb{E}[\mathcal{L}(w_T)] - \mathcal{L}^* \leq \max \left\{ \left(1 - \frac{\beta}{d}\mu\gamma(1 - \Delta(v, f)V)\right)^{T-1} (\mathcal{L}(w_1) - \mathcal{L}^* - H) + H, H \right\} \quad (54)$$

$$= \left(1 - \frac{\beta}{d}\mu\gamma(1 - \Delta(v, f)V)\right)^{T-1} (\mathcal{L}(w_1) - \mathcal{L}^* - H) + H \quad (55)$$

which concludes the proof.  $\square$

### 3.6. Proof of Theorem 2

**Theorem** (Non-convex convergence). *Suppose that Assumptions 1, 3 and 4 hold. If  $\gamma < \frac{1}{\sqrt{dL}}$  and  $\Delta(v, f)V < 1$  then*

$$\min_{k \in [T]} \mathbb{E} \left[ \|\nabla \mathcal{L}(w_k)\|^2 \right] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{T\beta\gamma(1 - \Delta(v, f)V)} + \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)}$$

where  $U = \max_{k \in [T]} U_{w_k}$ ,  $V = \max_{k \in [T]} V_{w_k}$  and  $\beta = \max \left\{ \tau \frac{\mu_\lambda}{\mu_s} \left( 1 - \sqrt{(\delta_\lambda^2 + \delta_s^2) \frac{v-1}{q-v+1}} \right), b_{\min} \right\}$ .

*Proof.* As in the proof of the previous theorem we decompose this proof into two parts:

**A).** All the elements of the sequence  $\{w_k\}_{k \in [T]}$  are in the subspace  $\mathcal{W}$ .

Let us recall that from (39), for any  $k \in [T]$  we have

$$\mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\mathbb{E}_{\mathcal{B}_k}[\|\mathcal{B}_k\|]}{2d} \gamma(1 - \Delta(v, f)V_{w_k}) \|\nabla \mathcal{L}(w_k)\|^2 + \frac{\gamma \Delta(v, f)U_{w_k}}{2}. \quad (56)$$

Let  $U = \max_{k \in [T]} U_{w_k}$ ,  $V = \max_{k \in [T]} V_{w_k}$ , and  $\beta = \min_{k \in [T]} \mathbb{E}_{\mathcal{B}_k}[\|\mathcal{B}_k\|]$ . Using these upper bounds, for any  $k \in [T]$  we get

$$\mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\beta \gamma}{2d} (1 - \Delta(v, f)V) \|\nabla \mathcal{L}(w_k)\|^2 + \frac{\gamma \Delta(v, f)U}{2}. \quad (57)$$

Rearranging the terms gives

$$\|\nabla \mathcal{L}(w_k)\|^2 \leq \frac{2d(\mathcal{L}(w_k) - \mathbb{E}_{\mathcal{B}_k} \mathbb{E}_k[\mathcal{L}(w_{k+1})])}{\beta \gamma(1 - \Delta(v, f)V)} + \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)} \quad (58)$$

As  $\mathbb{E}[\cdot] = \mathbb{E}_1[\dots \mathbb{E}_t[\cdot] \dots]$ , from above we obtain that

$$\mathbb{E} \|\nabla \mathcal{L}(w_k)\|^2 \leq \frac{2d(\mathbb{E}[\mathcal{L}(w_k)] - \mathbb{E}[\mathcal{L}(w_{k+1})])}{\beta \gamma(1 - \Delta(v, f)V)} + \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)} \quad (59)$$

As the above is true for all  $k \in [T]$ , we can sum both sides through  $[1, \dots, T]$  to get

$$\sum_{k=1}^T \mathbb{E} \|\nabla \mathcal{L}(w_k)\|^2 \leq \frac{2d(\mathbb{E}[\mathcal{L}(w_1)] - \mathbb{E}[\mathcal{L}(w_{T+1})])}{\beta \gamma(1 - \Delta(v, f)V)} + T \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)} \quad (60)$$

Dividing the last inequality by  $T$  gives:

$$\mathbb{E} \left[ \frac{1}{T} \sum_{k=1}^T \|\nabla \mathcal{L}(w_k)\|^2 \right] \leq \frac{2d(\mathcal{L}(w_1) - \mathbb{E}[\mathcal{L}(w_{T+1})])}{Tb\gamma(1 - \Delta(v, f)V)} + \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)} \quad (61)$$

And using the fact that  $\mathbb{E}[\mathcal{L}(w_{T+1})] > \mathcal{L}(w_*)$ :

$$\mathbb{E} \left[ \frac{1}{T} \sum_{k=1}^T \|\nabla \mathcal{L}(w_k)\|^2 \right] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{Tb\gamma(1 - \Delta(v, f)V)} + \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)}, \quad (62)$$

which also means that

$$\min_{k \in [T]} \mathbb{E} \left[ \|\nabla \mathcal{L}(w_k)\|^2 \right] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{kb\gamma(1 - \Delta(v, f)V)} + \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)}. \quad (63)$$

**B).** There exists at least one element  $w_s$  of the sequence  $\{w_k\}_{k \in [T]}$  in the subspace  $\mathbb{R}^d \setminus \mathcal{W}$ .

By construction of  $\mathcal{W}$ , as in (49), we have:

$$\|\nabla \mathcal{L}(w_s)\|^2 < \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)} \quad (64)$$

**A+B).** By combining (62) and (64), we obtain,  $\forall w \in \mathbb{R}^d$ :

which also means that:

$$\min_{1 \leq i \leq T} \mathbb{E} \left[ \|\nabla \mathcal{L}(w_i)\|^2 \right] \leq \max \left\{ \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{kb\gamma(1 - \Delta(v, f)V)} + \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)}, \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)} \right\} \quad (65)$$

$$= \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{kb\gamma(1 - \Delta(v, f)V)} + \frac{d\Delta(v, f)U}{\beta(1 - \Delta(v, f)V)} \quad (66)$$

and that concludes the proof.  $\square$

### 3.7. Proof of Corollary 1

**corollary.** Let  $\lambda_i$  be the computation rate of worker  $i \in \mathcal{C}$ , where  $\mathcal{C}$  denotes the set of correct workers. Let  $\tau > 0$  be the server's waiting time per iteration and  $\Delta = \Delta(v, f)$  be the robustness coefficient of a robust aggregation rule used in Algorithm 1. To ensure convergence, the average computation rate of correct workers  $\bar{\lambda}$  must satisfy the following:

$$\bar{\lambda} \geq \frac{d}{\tau} \left( \frac{1}{\sqrt{dQ_M\Delta}} - Q'_M \right)^{-1} = \Omega \left( \frac{\Delta d}{\tau} \right)$$

where  $\bar{\lambda} = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \lambda_i$ ,  $Q_M = \max_{w \in \mathbb{R}^d} Q_w$  and  $Q'_M = \max_{w \in \mathbb{R}^d} Q'_w$ .

*Proof.* Let  $Q_M = \max_{w \in \mathbb{R}^d} Q_w$  and  $Q'_M = \max_{w \in \mathbb{R}^d} Q'_w$ . We have then:

$$V = \max_{w \in \mathbb{R}^d} \left\{ \frac{1}{|\mathcal{C}|} \sqrt{\sum_{j=1}^d \left( \sum_{i \in \mathcal{C}} Q_w(Q'_{w_k} + \frac{1}{\lambda'_i}) \right)^2} \right\} \quad (67)$$

$$= \frac{1}{|\mathcal{C}|} \sqrt{d \left( \sum_{i \in \mathcal{C}} Q_M(Q'_M + \frac{1}{\lambda'_i}) \right)^2} \quad (68)$$

$$= \frac{1}{|\mathcal{C}|} \sqrt{dQ_M} \left( |\mathcal{C}| Q'_M + \sum_{i \in \mathcal{C}} \frac{1}{\lambda'_i} \right) \quad (69)$$

In theorems 1 and 2, we introduced the necessary condition for convergence  $\Delta V < 1$ . Substituting  $V$  as in Equation 69 gives:

$$\Delta \frac{1}{|\mathcal{C}|} \sqrt{dQ_M} \left( |\mathcal{C}| Q'_M + \sum_{i \in \mathcal{C}} \frac{1}{\lambda'_i} \right) \leq 1 \quad (70)$$

$$|\mathcal{C}| Q'_M + \sum_{i \in \mathcal{C}} \frac{1}{\lambda'_i} \leq \frac{|\mathcal{C}|}{\sqrt{dQ_M\Delta}} \quad (71)$$

$$\sum_{i \in \mathcal{C}} \frac{1}{\lambda'_i} \leq \frac{|\mathcal{C}|}{\sqrt{dQ_M\Delta}} - |\mathcal{C}| Q'_M \quad (72)$$

$$\frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \frac{1}{\lambda'_i} \leq \frac{1}{\sqrt{dQ_M\Delta}} - Q'_M \quad (73)$$

Since  $\lambda'_i = \frac{|\mathcal{S}_i| - 1}{\tau \lambda_i}$ , then we also have:

$$\frac{1}{\lambda'_i} = \frac{\frac{|\mathcal{S}_i|d}{\tau \lambda_i} - 1}{|\mathcal{S}_i| - 1} \quad (74)$$

$$= \frac{\frac{d}{\tau \lambda_i} - \frac{1}{|\mathcal{S}_i|}}{1 - \frac{1}{|\mathcal{S}_i|}} \geq \frac{d}{\tau \lambda_i} \quad (75)$$

And therefore:

$$\frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \frac{1}{\lambda'_i} \geq \frac{d}{\tau} \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \frac{1}{\lambda_i} \quad (76)$$



Using 76 in 73, we obtain:

$$\frac{1}{|C|} \sum_{i \in C} \frac{1}{\lambda_i} \leq \left( \frac{1}{\sqrt{d} Q_M \Delta} - Q'_M \right) \frac{\tau}{d} \quad (77)$$

$$\left( \frac{1}{|C|} \sum_{i \in C} \frac{1}{\lambda_i} \right)^{-1} \geq \left( \frac{1}{\sqrt{d} Q_M \Delta} - Q'_M \right)^{-1} \frac{d}{\tau} \quad (78)$$

Note that  $\left( \frac{1}{|C|} \sum_{i \in C} \frac{1}{\lambda_i} \right)^{-1}$  is the harmonic mean of  $\{\lambda_i | i \in C\}$ . Since arithmetic mean is always greater than harmonic mean, Inequality 78 becomes:

$$\frac{1}{|C|} \sum_{i \in C} \lambda_i \geq \left( \frac{1}{\sqrt{d} Q_M \Delta} - Q'_M \right)^{-1} \frac{d}{\tau} \quad (79)$$

which concludes the proof.  $\square$

#### 4. Additional details on experiments

In this work, we choose linear regression, logistic regression, support vector machine and neural networks as ML models to test the performance of several instances of ARGO against SGD. We also use the datasets, MNIST, BOSTON and PHISHING. In the following, we describe their properties as well as the pre-processing steps adopted for each one of them.

##### 4.1. ML models

*Linear regression.* . This linear model is used to predict a scalar value based on a set of explanatory variables. It is very efficient when the relation between the input features and the output is linear. The cost function in this case is a simple squared euclidean norm of the residuals:

$$C(w) = \|Xw - y\|^2$$

*Logistic regression.* . This model computes the probabilities for classification problems with two possible outcomes. Instead of using a simple linear hypothesis  $Xw$ , as in linear regression, a sigmoid function is applied to this linear hypothesis:  $h_w(X) = \frac{1}{1+e^{-Xw}}$ . Combined with the logistic loss, the cost function of this model can be written as follows:

$$C(w) = \frac{1}{m} \sum_{i=1}^m -y_i \log(h_w(X_i)) + (1 - y_i) \log(1 - h_w(X_i))$$

*Support vector machine.* . This last model constructs a hyperplane in a high-dimensional space that can be used for classification tasks. In this work, we choose the regularized version, and the cost function is written as follows:

$$C(w) = \frac{1}{2} \|w\|^2 + C \left[ \frac{1}{m} \sum_{i=1}^m \max(0, 1 - y_i(w x_i)) \right]$$

*Neural networks.* . Neural network learns to map a set of inputs to a set of outputs from training dataset. A deep neural network consists of an input layer representing the set of input features  $X$ , an output layer representing the number of classification classes, and a number of hidden layers with multiple hidden units, as depicted in Figure 3. Each unit computes its value based on the combination of values from previous layers and an activation function. The most common activation functions are:

- Sigmoid:  $\sigma(z) = \frac{1}{1+e^{(-z)}}$
- Tanh:  $\tanh z = \frac{e^{(z)} - e^{(-z)}}{e^{(z)} + e^{(-z)}}$
- ReLU (Rectified Linear Unit):  $ReLU(z) = \max(0, z)$

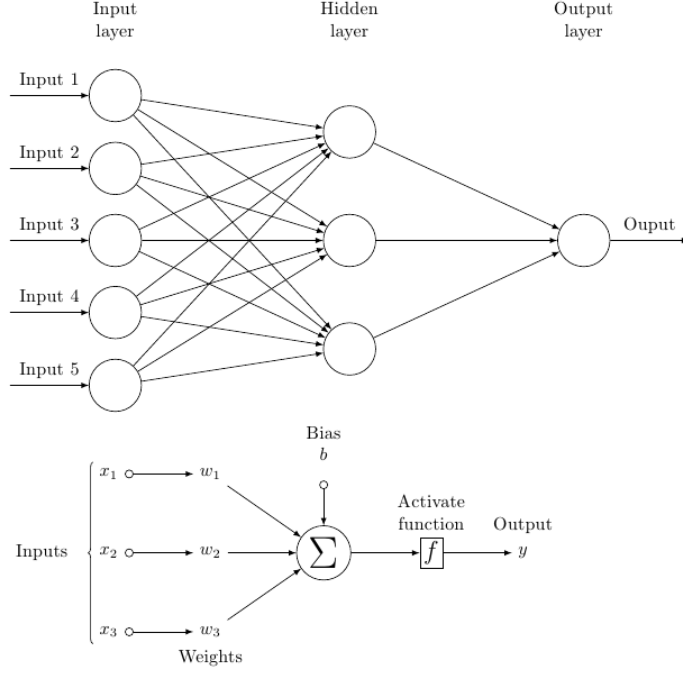


Figure 3: A feed-forward neural network with a 5-inputs layer, one hidden layer with 3 neurons and one output layer with only one class. Each neuron is composed of a transfer function summing all the inputs and a bias, followed by an activation function.

#### 4.2. Datasets and pre-processing

To conduct an extensive and diversified set of experiments, we have considered multiple datasets. Each dataset is used with a specific ML model.

**MNIST dataset.** The MNIST dataset consists in 10 categories of digits, from 0 to 9, represented by a set of 784 features, with a total of 70,000 data samples (60,000 for training and 10,000 for testing). The training set is shuffled and normalized; then, each worker is assigned a non-overlapping i.i.d. sample from the training set, drawn uniformly. MNIST is used with logistic regression to perform a binary classification, and neural networks for a full classification.

**BOSTON dataset.** The BOSTON dataset contains information (collected by the US Census Service) about housing in the area of Boston (Massachusetts). The dataset contains 506 data samples and 14 feature variables. After standardizing its features, each worker is assigned (randomly) a non-overlapping i.i.d. sample from the training set. The dataset is used in conjunction with a linear regression model to predict the price of houses.

**PHISHING dataset.** The PHISHING dataset contains fraudulent attempts to obtain sensitive information (e.g., usernames, passwords, and credit card details) through email spoofing, instant messaging and text messaging. The dataset contains 8844 fraudulent attempts, characterized by 68 features. All features have been scaled by their maximum absolute value and normalized. Then, each worker is assigned (randomly) a non-overlapping i.i.d. sample from the training set. The dataset is used with a support vector machine (SVM) model to check whether the URL is phishing or not.

In the experiments involving non identically distributed datasets, each worker will only be receiving a random sample of the training set, containing only a few classes.

#### 4.3. Specification of experiments

**Block generation for neural networks.** In the case of neural networks, block generation is different. As a matter of fact, stacking all the weights into a single vector and picking randomly a block constructed as defined above will result in an unbalanced updates between the network layers. Instead, we use a vector containing the number of coordinates (weights) to be updated in each layer. For example, in a feed-forward neural network with 784 inputs, one hidden layer with 30 neurons and an output with 10 classes (a total of 3 layers), a block size will be written in the following format: [98,20,10]. This means that 98 out of 784 weights in the first layer, 20 out of 30 weights in the second layer and all the weights in the last layer are selected randomly and updated at each round.

**Runs.** Because of the stochastic nature of the algorithms, each experiment is run 10 times. We compute the mean and the standard deviation of the metric used in each experiment, and we plot the mean as well as a confidence interval.

**Metrics.** For the linear regression model, the evaluation metric is the cost function. For the classification tasks using either logistic regression, support vector machine or neural networks, the evaluation metric is accuracy.

**Computation rates and server waiting time.** In this work, we introduced the computation rate variable  $\lambda$  to model the computing capabilities of workers. In a real world deployment of ARGO, the computation rate of a device could be the frequency of its processor, the number of cores it has, the RAM capacity, the LTE speed, etc..., or a combination of these measures. Since we are only running simulations of a distributed network of mobile devices, we only choose a random value of the computation rate of each computation profile (weak, average, powerful), then each worker is assigned a random computation rate following normal distribution whose mean is equal to the computation rate randomly chosen, depending on the computation profile category. Of course, the values of weak computation rates must satisfy the fact weak workers need at least to be able to compute one partial derivative. Beside the computation rate of workers, the server has also an impact on the number of partial derivatives a worker can produce, by varying the waiting time  $\tau$ . In our experiments, we set this variable so that weak workers are able to produce at least one partial derivative.

#### 4.4. Attacks used in the Byzantine experiments

It is known that AVERAGE is not a robust aggregation rule. In fact, as shown in [7], one single Byzantine worker is sufficient to give any value to the aggregated gradient. Moreover, without any computation efforts, a Byzantine worker can send very large coordinate values. Using this simple strategy can make the ML models diverge. In this work, we used two state-of-the-art attacks that were developed against the most effective robust variants of SGD. In the plots, we always include the convergence graph of ARGO with AVERAGE under no attack, as a benchmark to assess the impact of Byzantine workers on the convergence behavior. For the sake of comparison, we also include ARGO with the AVERAGE aggregation rule as a reference. In some experiments, this version shows a good performance against those attacks. However, this does not mean that AVERAGE is Byzantine-resilient, and practitioners are aware of it because of proven vulnerabilities.

**A little is enough.** This attack was presented in [8]. The idea consists in sending Byzantine values that are not too far from the mean, and can be confused with correct values if we consider a normal distribution of correct values. In other words, the Byzantine worker will send values that are far from the mean within an interval of  $z$  standard deviations. Algorithm 1 summarizes the attack.

---

#### Algorithm 1 “A LITTLE IS ENOUGH” ATTACK

---

**Input:**  $f, n$   
 $n = \lfloor \frac{n}{2} + 1 \rfloor - f$   
 $z_{max} = \max_z \left( \phi(z) < \frac{n-m-s}{n-m} \right)$   
**for**  $j = 1$  to  $d$  **do**  
    calculate mean ( $\mu_j$ ) and standard deviation ( $\sigma_j$ )  
     $[B]_j = \mu_j + z_{max}\sigma_j$   
**end for**  
**for**  $i = 1$  to  $f$  **do**  
     $B_i = B$   
**end for**

---

**Fall of empires.** This attack was presented in [9]. This attack tries to manipulate the inner product of the true gradient and the aggregated vector by making it negative. In order to guarantee the progress of any descent algorithm, this inner product must stay positive. The attack is very simple and consists in sending the negative of correct vectors’ average, multiplied by a value  $\epsilon$ . Formally, if  $\mathcal{C}$  is the set of correct workers, then:

$$\forall j \in [f], \quad B_j = -\frac{\epsilon}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} G_i$$

#### 4.5. Hardware, libraries and dependencies

All experiments were conducted on a MacBook Pro (2018) with a 2.3 GHz Quad-Core Intel Core i5 processor and 16.0G of RAM. The source code was written in Python (version 3.8) and makes use of usual machine learning libraries, namely: Numpy, Pandas and Scikit-learn.

#### 4.6. Source code

The source code for all experiments is available in the following GitHub repositories:

- Distributed implementation:  
<https://github.com/karimboubouh/Hg0>
- Android application:  
<https://github.com/karimboubouh/Hg0App>

More details are provided in the repositories on how to reproduce the results of the paper.

## 5. ARGO: Android application

In the previous section, we implemented ARGO in a simulated distributed environment (on a computer), and evaluated all convergence properties, namely: accuracy, iteration cost, convergence speed and Byzantine resilience. In this section, we report on the deployment of our algorithm as an Android application. As opposed to the simulations, where workers and servers are represented as classes in object-oriented programming, this experiment is a truly distributed training where workers are physically distinct (smartphones) and are connected via WiFi (local network) to a parameter server (PS) hosted in a computer. Since the aggregation at the PS is also lightweight, the PS can be hosted in one of the smartphones involved in the training. This app was developed to test ARGO in a network with smartphones, and show that ML is actually possible using these devices. Therefore, the user interface and the app capabilities are very limited, with only essential functionalities to prove our points. A full version of the app is left for future work.

### 5.1. Setup

The smartphone implementation of ARGO uses the python cross-platform library Kivy (<https://kivy.org>) and is optimized to deploy the Android version<sup>1</sup> of ARGO. The application works on Android version 7+, and only requires storage permission to access the smartphone local dataset. The parameter sever handles the connection and the disconnection of devices transparently, and can be hosted in a local network or at any public IP address.

### 5.2. Tutorials

To join the distributed training from the Android application, the application first needs to be configured (Figure 4a, right side) by setting the correct IP address and port of the server (the user must make sure they are in the same network, in case of a private IP address). Next, the computation profile needs to be selected. For simplicity, we suggest three configurable computation profiles with the following default values:

- Low configuration: implying a low computation rate.
- Moderate configuration: implying an average computation rate.
- Powerful configuration: no assumption on the computation rate

Once configured, the application will receive the selected model from the PS. The PS will wait for enough devices to join. Then, it sends the current model parameters and the selected block to train in the next round. Once the training is finished, a new screen is activated with a summary of the training.

## References

- [1] S. J. Wright, Coordinate descent algorithms, *Mathematical Programming* 151 (1) (2015) 3–34.
- [2] C. Xie, O. Koyejo, I. Gupta, Phocas: dimensional byzantine-resilient stochastic gradient descent (2018). *arXiv*: 1805.09682.
- [3] E. Díaz-Francés, F. Rubio, On the existence of a normal approximation to the distribution of the ratio of two independent normal random variables, *Statistical Papers* 54 (05 2013). doi:10.1007/s00362-012-0429-2.
- [4] B. Arnold, N. Balakrishnan, Relations, Bounds and Approximations for Order Statistics, *Lecture Notes in Statistics*, Springer New York, 2012.  
URL <https://books.google.co.ma/books?id=Tb7kBwAAQBAJ>
- [5] A. Mood, *Introduction to the Theory of Statistics*, McGraw-Hill, 1950.
- [6] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, 1st Edition, Springer Publishing Company, Incorporated, 2014.
- [7] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, J. Stainer, Machine learning with adversaries: Byzantine tolerant gradient descent, *NeurIPS’17*, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 118–128.

---

<sup>1</sup>The application is cross-platform and can also be deployed on IOS, Linux, MacOS and Windows.

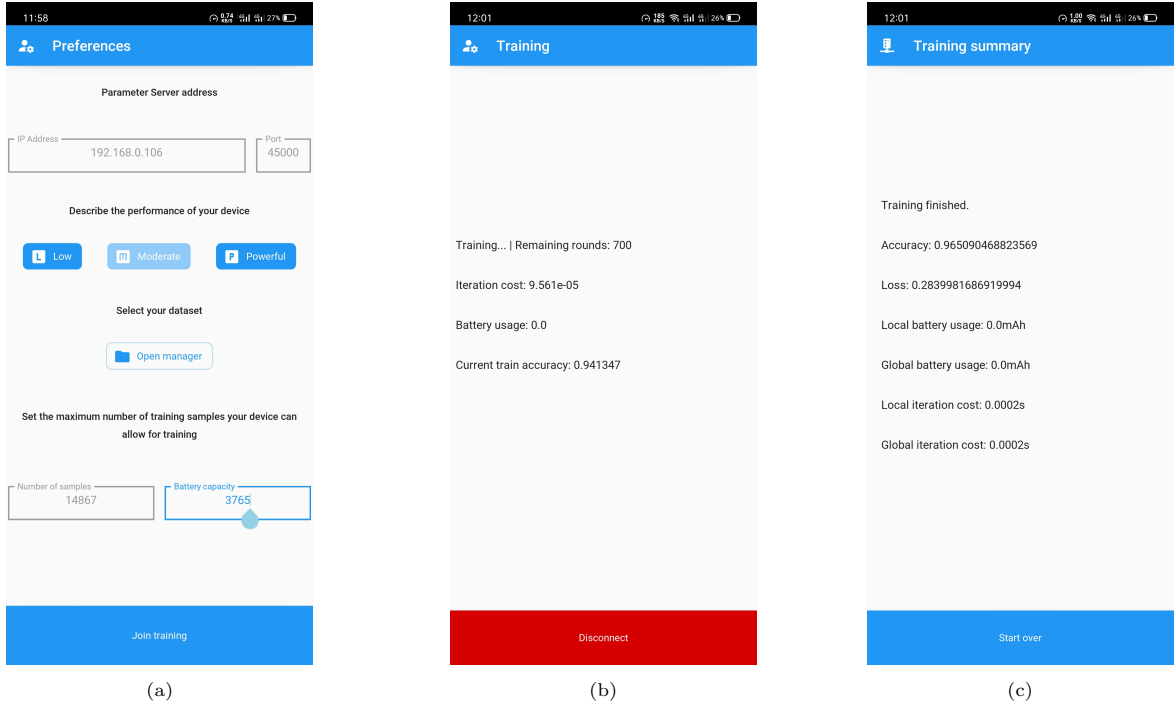


Figure 4: ARGO running on an Android smartphone. The left side figure shows the configuration screen of the application. The middle figure shows the app status during the training, and the right side figure shows the final screen, when the training is complete.

- [8] G. Baruch, M. Baruch, Y. Goldberg, A little is enough: Circumventing defenses for distributed learning, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 32, Curran Associates, Inc., 2019, pp. 8635–8645.  
URL <https://proceedings.neurips.cc/paper/2019/file/ec1c59141046cd1866bbbcdfb6ae31d4-Paper.pdf>
- [9] C. Xie, S. Koyejo, I. Gupta, Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation (2019).  
arXiv:1903.03936.

# Democratizing Machine Learning

Resilient Distributed Learning with Heterogeneous Participants

Karim Boubouh

UM6P

Benguerir, Morocco

karim.boubouh@um6p.ma

Amine Boussetta\*

UM6P

Benguerir, Morocco

amine.boussetta@um6p.ma

Nirupam Gupta

EPFL

Lausanne, Switzerland

nirupam.gupta@epfl.ch

Alexandre Maurer

UM6P

Benguerir, Morocco

alexandre.maurer@um6p.ma

Rafaël Pinot

EPFL

Lausanne, Switzerland

rafael.pinot@epfl.ch

**Abstract**—The increasing prevalence of personal devices motivates the design of algorithms that can leverage their computing power, together with the data they generate, in order to build privacy-preserving and effective machine learning models. However, traditional distributed learning algorithms impose a uniform workload on all participating devices, most often discarding the weakest participants. This not only induces a suboptimal use of available computational resources, but also significantly reduces the quality of the learning process, as data held by the slowest devices is discarded from the procedure.

This paper proposes HGO, a distributed learning scheme with parameterizable iteration costs that can be adjusted to the computational capabilities of different devices. HGO encourages the participation of slower devices, thereby improving the accuracy of the model when the participants do not share the same dataset. When combined with a robust aggregation rule, HGO can tolerate some level of Byzantine behavior, depending on the hardware profile of the devices (we prove, for the first time, a trade-off between Byzantine tolerance and hardware heterogeneity). We also demonstrate the convergence of HGO, theoretically and empirically, without assuming any specific partitioning of the data over the devices. We present an exhaustive set of experiments, evaluating the performance of HGO on several classification tasks and highlighting the importance of incorporating slow devices when learning in a Byzantine-prone environment with heterogeneous participants.

**Index Terms**—Distributed computing, Byzantine failures, Heterogeneity, Machine learning, Optimization

## I. INTRODUCTION

Mobile devices are now an essential component of our modern society. Their number of users will reach 5.1 billion by 2025, representing 70% of the world’s population.<sup>1</sup> Each day, these devices generate a massive amount of data. Rather than letting intermediary companies exploit the data, it is appealing to have each device locally process the data it has generated, both for infrastructure cost efficiency and privacy reasons. State-of-the-art distributed machine learning schemes enable

devices (usually called workers) to collaboratively learn a common model by exchanging information about the data, possibly with the help of a machine server [1], [2]. However, these schemes typically assume participating devices with similar hardware specifications, thereby imposing the same workload on every worker. This requirement might be satisfied in data centers equipped with powerful co-located computers, steadily connected to the power grid and to the network. However, personal computing devices owned by the general public, such as smartphones or laptops, often behave in an unpredictable way. They may disconnect from the network very often, become frequently inactive to save battery consumption, or crash accidentally. Besides, some devices, technically referred to as *Byzantine* [3], might get hacked and manipulated by malicious parties with the goal of disrupting the system. Most importantly, most of these portable devices cannot handle the iteration complexity (computing a full gradient) of first-order optimization algorithms (i.e. SGD [4]) applied on very complex models (i.e. GPT3 [5], with 175 billion parameters).

Traditional distributed machine learning algorithms are ill-suited for such handheld devices. First, synchronous algorithms usually show a stable convergence, but the waiting time is dictated by the weakest device. On the other hand, asynchronous variants are indeed faster, but they induce high variance in the updates, leading to poor solutions [6]. One can accelerate the training simply by discarding weaker devices (with low computational power) from the learning process. Whilst this usually has little impact when all workers share the same dataset (a.k.a. the homogeneous case), it can significantly impact the quality of the learning with heterogeneous data. Finally, it should be noted that these traditional algorithms assume that workers are able to compute a full gradient, which is clearly a strong assumption in the case of smartphone-based machine learning.

In short, three critical issues need to be considered when deploying distributed ML algorithms on handheld devices: the *computational heterogeneity* of the devices, the *heterogeneity of their datasets*, and the possible *Byzantine behavior* of a

\* Corresponding author.

<sup>1</sup>According to the Mobile Economy 2019 report by the GSM Association: <https://data.gsmainelligence.com/api-web/v2/research-file-download?id=39256194&file=2712-250219-ME-Global.pdf>

fraction of them. In this paper, we present HGO, a new distributed learning protocol that addresses all of these three issues. More specifically, our technical contribution is twofold.

- 1) We present a non-trivial optimization protocol, called HGO, that adapts the computational workload to the capabilities of the devices, tackling (directly) computational heterogeneity and (indirectly) data heterogeneity, resulting in a more efficient use of available computing resources and improved learning accuracy.
- 2) We provide theoretical and empirical evidence that HGO can learn accurate models even in a demanding environment that combines heterogeneous and Byzantine participants. In particular, our results highlight the importance of incorporating slow devices when learning in this challenging environment, thus demonstrating the superiority of HGO over traditional schemes such as SGD in this context.

For presentation simplicity, and to focus on HGO’s originality, in this paper, we do not discuss the problem of tolerating a Byzantine parameter server, which we assume to be unique and trusted [1], [2], [7] (a very common assumption in Byzantine machine learning [8]–[15]). However, note that HGO can be used with the replication scheme of [16] to circumvent that assumption.

**Overview of the protocol.** Essentially, HGO can be viewed as a robust, heterogeneity-aware variant of a stochastic *block coordinate descent* algorithm. The server initiates each iteration by randomly choosing a subset of  $q$  workers, and sends them its current estimates of the learning parameters. Every honest (non-Byzantine) worker responds with a fragment (i.e., some coordinates) of its mini-batch stochastic gradient. Upon receiving these information, the server identifies a block of coordinates that have been computed by a sufficiently large quorum of at least  $v$  workers, and applies a robust aggregation rule on this block to update its current parameter estimates. The Byzantine tolerance of HGO depends on the underlying aggregation rule. We use *Trimmed Mean* [11] as a default aggregation rule, tolerating up to  $f < \frac{v}{2}$  Byzantine workers. However, note that any coordinate-wise robust aggregation rule could be used instead.

**Analysis and benefits.** We prove linear and sub-linear convergence of HGO in the strongly convex and non-convex settings, respectively. We also study its learning performance empirically on several classification tasks. In case our solution is deployed using actual smartphones, we developed an application running on Android operating systems. We include a quick tutorial on how to setup the application in the appendix, as well as in the GitHub repository (link in Section VI). Our key theoretical and empirical findings demonstrate that, maybe surprisingly, it is possible to incorporate slow devices in the learning procedure without slowing down the protocol by de-correlating the workload of the workers from the computational power of the strongest devices. We also demonstrate that improving the slow device participation considerably outperforms traditional schemes in terms of final

accuracy of the model, when the dataset is not shared by all the workers. This is achieved without hampering the Byzantine resilience of the process.

Our protocol and its analysis can be applied to a large family of gradient-based optimization schemes (including CD [17], SCD [18], Block CD [19], GD [20] and SGD [4]) as well as to a large set of robust aggregation rules (including Trimmed Mean [11], Aksel [9] and MeaMed [21]). To the best of our knowledge, none of these learning algorithms has been analyzed under the combination of the three major issues we consider: *computational heterogeneity*, *data heterogeneity*, and *Byzantine behavior*. Furthermore, HGO is fairly general, and could easily be extended with more advanced solvers, such as NAG [22], RMSprop [23], AdaGrad [24], Adam [25], etc.

Before going further, we would like to stress the fact that this paper proposes a new solution to computational heterogeneity, by allowing workers to compute only fractions of gradients, depending on their hardware capabilities. We do not claim novelty on Byzantine resilience nor data heterogeneity, which are classical subjects in this line of research. Instead, we combine our new scheme with robust aggregation rules into one algorithm, which we called HGO, and we analyze its convergence properties under Byzantine attacks, assuming computational heterogeneity among workers, as well as heterogeneity among local datasets, whose level is estimated based on the bias value of the estimation of partial derivatives. Beside the novelty of the solution itself, we believe we are the first to introduce computational heterogeneity in the convergence analysis of gradient-based optimization algorithms. In particular, we demonstrate, for the first time, a critical tension between the computational capabilities of the workers and the impact Byzantine workers can have on the final accuracy of the model. Basically, the more weak devices we have, the more Byzantine workers can affect the convergence properties.

**Roadmap.** The rest of the paper is organized as follows. We first discuss some related works in Section II, and explain how our solution is different from previous works on computational heterogeneity. We then present the model and preliminary concepts in Section III. Section IV describes our HGO algorithm and its time complexity. Section V presents our theoretical analysis of HGO and highlights some interesting trade-offs. Section VI presents a selection of experiments demonstrating the practical relevance of HGO, and Section VII concludes the paper. For space limitations, we defer all the proofs, as well as an exhaustive set of experiments, to an appendix. We also provide an open-source Android implementation of HGO, available at: <https://github.com/karimboubouh/HgOApp>.

## II. RELATED WORK

Prior works in distributed ML consider the issues of heterogeneous computational capabilities, heterogeneous datasets and Byzantine behavior separately. For example, several approaches studied the Byzantine resilience of gradient descent optimization algorithms [8]–[15] by leveraging robust statistics, filtering schemes or coding theory, to defend against a fraction of Byzantine workers. However, they all assumed



homogeneous data distribution and computational capabilities. Some other works [26]–[28] studied SGD with local iterations on heterogeneous data, without assuming Byzantine and computationally heterogeneous workers. Distributed SGD was also analyzed in [29], [30] assuming data heterogeneity and Byzantine attacks simultaneously, but computational heterogeneity was not addressed.

On the other hand, most works [10], [31], [32] claiming computational heterogeneity awareness do not question the ability of workers to compute a full gradient. Instead, the problem is only seen through the windows of asynchrony, meaning that workers are assumed to be capable of computing full gradient but may be stale. Most of these solutions propose either new synchronization schedules or new adaptive learning rate schedules, or a combination of both to tackle the problem of stragglers. Also note that, while compression schemes [33]–[36] were proven efficient in reducing communication complexity, they do not address computational heterogeneity.

While previously cited work clearly fail when workers are not able to compute a full gradient, our protocol provably converges under the same assumption. Basically, the server exploits information from every worker, using either full gradients, partial gradients or even one partial derivative. Most importantly, our theoretical analysis suggests new conditions for convergence involving the Byzantine impact, the heterogeneity level of local datasets and, for the first time, the computational capabilities of workers.

### III. PRELIMINARIES

We consider the parameter-server architecture [7], where a server orchestrates the training of a machine learning model with the help of a set of  $n$  workers. In this setup, the dataset is composed of  $N$  data points  $X = \{x_1, \dots, x_N\}$ , arbitrarily partitioned over the  $n$  workers. The set of data points held by a worker  $i$  is denoted by  $\mathcal{S}_i$ . Thus,  $\bigcup_{i=1}^n \mathcal{S}_i = X$ , and  $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$  for all  $i \neq j$ . The server is assumed to follow the algorithm correctly, but some of the workers may be faulty, and their identity is a priori unknown [3]. More specifically, there are two types of workers:

- 1) *Correct workers*: These participants, a.k.a. *honest workers*, correctly follow the instructions prescribed by the server. We represent the correct workers by the set  $\mathcal{C} \subseteq [n]$ , where  $[n]$  denotes the set  $\{1, \dots, n\}$ .
- 2) *Byzantine workers*: Workers that are not honest, i.e.,  $[n] \setminus \mathcal{C}$ , are assumed to be Byzantine. This means that they have an arbitrary behavior and can either crash or inject adversarial perturbations in the system (see, e.g., [8]).

Our goal is to design a distributed learning algorithm that allows all correct workers to learn a model over the union of their data points  $X_{\mathcal{C}} = \bigcup_{i \in \mathcal{C}} \mathcal{S}_i$ , despite the presence of up to  $f$  Byzantine workers. Specifically, we fix a learning model parameterized by  $w \in \mathbb{R}^d$ , for which each data point  $x$  incurs a loss of  $\ell(w, x)$ , where the function  $\ell : \mathbb{R}^d \times X \rightarrow \mathbb{R}$  is assumed

differentiable. The algorithm aims to find a local minimum of the following function:

$$\mathcal{L}(w) = \frac{1}{|X_{\mathcal{C}}|} \sum_{x \in X_{\mathcal{C}}} \ell(w, x) \quad (1)$$

Besides the presence of faulty workers, another challenge we consider in our model is the heterogeneous computational capabilities of the workers. To formally discuss this issue, we first briefly describe the standard distributed stochastic gradient descent (SGD) scheme below.

#### A. Distributed SGD

Distributed SGD is an iterative algorithm where the server maintains a parameter estimate, which is updated iteratively with the help of the workers. We assume that time proceeds in discrete steps, and that each part of the algorithm's execution is allocated a given amount of time units. The initial estimate  $w_1$  may be chosen arbitrarily. In each iteration  $k \geq 1$ , the server broadcasts its current estimate  $w_k$  to all workers, and waits during a fixed amount of time units for the workers to return their stochastic gradients computed at  $w_k$ . Each worker  $i$  samples a random *mini-batch*  $\zeta_{i_k} \subset \mathcal{S}_i$  of size  $s_i$  to compute the *stochastic gradient*:

$$G_i(w_k) = \frac{1}{s_i} \sum_{x \in \zeta_{i_k}} \nabla \ell(w_k, x) \quad (2)$$

Then, the workers send back their stochastic gradients to the server, which updates the model parameters as follows:

$$w_{k+1} = w_k - \gamma \left( \frac{1}{n} \sum_{i=1}^n G_i(w_k) \right) \quad (3)$$

where  $\gamma > 0$  is the learning rate. The algorithm can be proven to eventually output a solution to the learning problem [4]. However, the convergence of Distributed SGD often relies on the assumption that all workers have comparable (homogeneous) computational capabilities and hardware specifications, which is unrealistic when deploying distributed learning on low-powered hand-held devices, such as tablets or smartphones.

#### B. Device heterogeneity: computational power and data

We consider a setting where all devices might not complete the computation of their gradient estimate in the same numbers of time units. Given a parameter  $w$  and a data point  $x \in \mathcal{S}_i$ , we assume that every worker will compute its gradient coordinate by coordinate, at its own pace. Specifically, every worker  $i$  can compute  $\lambda_i > 0$  coordinates of gradient  $\nabla \ell(w, x)$  per time units. Then, in a given time frame of  $\tau$  time units, worker  $i$  can compute  $\min\{\frac{\lambda_i \tau}{s_i}, d\}$  coordinates of its stochastic gradient. We call  $\lambda_i$  the computational capability of worker  $i$ .

Then, for a fixed mini-batch size, slower workers (with lower computational capabilities) may only be able to compute their stochastic gradients partially, while the faster workers (with higher computational capabilities) may compute all the coordinates of their stochastic gradients. In what follows, we

denote by  $\mathcal{Q}_j$  the set of workers that computed coordinate  $j$  during a given time frame.

Besides the differences in workers' computational capabilities, our setting faces *data heterogeneity*. Indeed, the data partitioning among the workers may not be uniform. Thus, even the correct workers are unable to compute unbiased estimates of the true gradient  $\nabla \mathcal{L}(w)$ . This motivates the introduction of Assumptions 3 and 4 in Section III-D.

### C. Robust aggregation

In the presence of Byzantine workers, the server cannot rely on the simple averaging of all workers' gradients, as in (3). It should rather use a coordinate-wise robust *aggregation rule* AR, which we formally define below. In the following, for any vector  $u \in \mathbb{R}^d$ , its  $j$ -th coordinate is denoted by  $\{u\}_j$ .

**Definition 1.** For a given  $\rho \in [n]$ ,  $w \in \mathbb{R}^d$ , a real-valued aggregation rule AR is said to be  $\Delta(\rho, f)$ -robust if, for every subset  $\mathcal{Q} \subseteq [n]$  with  $|\mathcal{Q}| \geq \rho$  and for every  $j \in [d]$ , we have

$$\mathbb{E} \left[ |\{AG_w\}_j - \{\nabla \mathcal{L}(w)\}_j|^2 \right] \leq \Delta(\rho, f) \left( \frac{1}{|\mathcal{C} \cap \mathcal{Q}|} \sum_{i \in \mathcal{C} \cap \mathcal{Q}} \sigma_{ij}^2(w) \right)$$

where  $\{AG_w\}_j = \text{AR}(\{G_i(w)\}_j; i \in \mathcal{Q})$  and  $\sigma_{ij}^2(w) = \mathbb{E} \left[ |\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j|^2 \right]$ . The robustness coefficient  $\Delta(\rho, f)$  is a bounded positive real value that depends on  $\rho$  and  $f$ .

Our theoretical analysis applies for all aggregation rules satisfying Definition 1, as soon as the robustness coefficient  $\Delta(\rho, f)$  is sufficiently well behaved: simply put, for a fixed number of Byzantine workers  $f$ , the aggregation error should at least be constant, if not decreasing, when the total number of workers  $\rho$  grows. This includes a large range of aggregation rules such as *Phocas* [37], *Trimmed Mean* [11] or *AKSEL* [9]. We demonstrate in Proposition 1 (in the Appendix) that Trimmed Mean is an example of aggregation rules satisfying Definition 1.

### D. Assumptions

To formally analyze the convergence of our algorithm, we make the following standard assumption on the smoothness of the loss function [38]. We recall that  $\{u\}_j$  denotes the  $j$ -th element of vector  $u$ .

**Assumption 1** (Smoothness). For all  $j \in [d]$ , there exists  $L_j < \infty$  such that, for all  $w, w' \in \mathbb{R}^d$ ,  $|\{\nabla \mathcal{L}(w')\}_j - \{\nabla \mathcal{L}(w)\}_j| \leq L_j \|w' - w\|$ .

Although most of our results do not rely on the convexity assumption, we also present a convergence result when the loss function is strongly convex, as stated below.

**Assumption 2** (Strong convexity). There exists  $0 < \mu < \infty$  such that, for all  $w, w' \in \mathbb{R}^d$ ,  $\mathcal{L}(w') \geq \mathcal{L}(w) + \langle \nabla \mathcal{L}(w), w' - w \rangle + \frac{\mu}{2} \|w' - w\|^2$ , where  $\langle \cdot, \cdot \rangle$  denotes the inner product.

To analyze the convergence of HGO under data heterogeneity, we also make the following two assumptions. Note that we

do not rely on the (stronger) assumption of uniformly bounded variance, which is indeed quite difficult (and perhaps even impossible) to satisfy in the case of strong convexity [39]. Instead, we follow [38] and assume a non-uniform bound, as stated below in Assumption 4.

**Assumption 3** (Bounded bias). For all  $i \in [n]$  and  $j \in [d]$ , there exist  $\delta_{i,j} < \infty$  such that, for all  $w \in \mathbb{R}^d$ ,  $|\mathbb{E}_{x \sim \mathcal{S}_i} [\{\nabla \ell(w, x)\}_j] - \{\nabla \mathcal{L}(w)\}_j| \leq \delta_{i,j}$  where  $x \sim \mathcal{S}_i$  denotes a uniform sampling of  $x$  in  $\mathcal{S}_i$ .

**Assumption 4** (Non-uniform variance). For all  $w \in \mathbb{R}^d$ , there exists  $M_w < \infty$  and  $Q_w < \infty$  such that, for all  $i \in [n]$  and  $j \in [d]$ ,  $\mathbb{E}_{x \sim \mathcal{S}_i} \left[ (\{\nabla \ell(w, x)\}_j - \mathbb{E}_{x \sim \mathcal{S}_i} [\{\nabla \ell(w, x)\}_j])^2 \right] \leq M_w + Q_w \|\{\nabla \mathcal{L}(w)\}_j\|^2$ .

## IV. HETEROGENEOUS GRADIENT-BASED OPTIMIZATION

We now present our protocol called **Heterogeneous gradient-based Optimization (HGO)**.

### A. Overview of the protocol

The protocol, summarized in Algorithm 1, follows  $K$  steps of the following iterative procedure.

- 1) At each iteration  $k \in [K]$ , the server maintains a parameter vector  $w_k$ . To update its vector, the server chooses a random order of coordinate indices  $C_k$  (i.e.,  $C_k$  is a permutation of the set  $\{1, \dots, d\}$ ), selects a random set of  $q$  workers, and broadcasts  $(w_k, C_k)$  to these workers. Then, the server waits for  $\tau$  time units for workers' responses.
- 2) Upon receiving the broadcast message from the server, an honest worker  $i$  randomly samples a mini-batch of size  $s_i$  from its local data, and sends back to the server the first  $b_i = \min\{\frac{\tau \lambda_i}{s_i}, d\}$  coordinates in  $C_k$  of its stochastic gradient  $G_i(w_k)$ , as defined in (2) above. Note that a faster worker (with higher computational capabilities) may compute more coordinates of its stochastic gradient compared to a slower worker. Byzantine workers may provide arbitrary values for their partial derivatives.
- 3) After  $\tau$  time units, the server identifies the set of indices  $\mathcal{B}_k \subseteq C_k$  that have been computed by at least  $v$  workers out of the  $q$  selected workers (the procedure for this is described in steps 8, 9 and 10 of Algorithm 1). The server then applies a robust *aggregation rule* AR on the selected coordinates of the gradients sent by the workers, and uses its output to update the current parameter vector  $w_k$  (step 13 in Algorithm 1).

### B. Time complexity.

The standard distributed SGD algorithm aggregates the full gradients computed by all  $n$  workers, while HGO only aggregates truncated gradients computed by  $q < n$  random workers. Accordingly, HGO is faster than SGD when comparing these steps. Furthermore, while SGD's per-iteration time is dictated by the slowest worker, which means that the server waits for

---

**Algorithm 1** Heterogeneous gradient-based Optimization  
**Parameter Server**


---

- 1: **Input:**  $q$ , AR (aggregation rule),  $w_1, v, K, \gamma$
- 2: **Execute the following instructions for each step**  $k = 1, \dots, K$
- 3:  $C_k \leftarrow$  random permutation of  $\{1, \dots, d\}$
- 4: Select  $q$  random workers  $\{i_1, \dots, i_q\} \subseteq \{1, \dots, n\}$
- 5: Broadcast  $(w_k, C_k)$  to the selected  $q$  workers
- 6: Wait  $\tau$  time units to receive workers' gradients
- 7:  $\nabla_{i_1}, \dots, \nabla_{i_q} \leftarrow$  partial derivatives sent by the  $q$  workers, i.e., elements of their gradients at  $w_k$
- 8:  $R \leftarrow$  sort  $(|\nabla_{i_1}|, \dots, |\nabla_{i_q}|, \text{decreasing order})$
- 9:  $b_v \leftarrow v^{\text{th}}$  item in the list  $R$
- 10:  $\mathcal{B}_k \leftarrow$  first  $b_v$  elements in  $C_k$
- 11:  $\forall j \in \mathcal{B}_k$ , identify the set  $\mathcal{Q}_j \subseteq \{i_1, \dots, i_q\}$  of workers that sent the  $j$ -th coordinate of their gradients.
- 12: Compute the aggregate of workers' partial derivatives  $\text{AG}_{w_k} \in \mathbb{R}^d$  such that, for all  $j \in [d]$ ,

$$\{\text{AG}_{w_k}\}_j = \begin{cases} \text{AR}(\{\nabla_i\}_j, i \in \mathcal{Q}_j) & \text{if } j \in \mathcal{B}_k \\ 0 & \text{otherwise} \end{cases}$$

where  $\{\nabla_i\}_j$  is sent by worker  $i$  for the  $j$ -th element of its gradient.

- 13: Update the parameter vector:  $w_{k+1} = w_k - \gamma \text{AG}_{w_k}$

---

**Honest Worker  $i$** 


---

- 1: **Input:** local dataset  $\mathcal{S}_i$ , mini-batch size  $s_i$ , computational capability  $\lambda_i$
  - 2: **If** the broadcast message  $(w_k, C_k)$  is received **then** execute the following steps:
  - 3: Sample a random mini-batch  $\zeta_{i_k}$  of size  $s_i$  from dataset  $\mathcal{S}_i$
  - 4:  $\mathcal{B}_k^i \leftarrow$  first  $b_i$  elements in  $C_k$ , where  $b_i = \min\{\frac{\tau \lambda_i}{s_i}, d\}$
  - 5: Construct the set  $\nabla_i = (\{G_i(w_k)\}_j, j \in \mathcal{B}_k^i)$ , where  $G_i(w_k)$  is as defined in (2)
  - 6: Send back  $\nabla_i$  to the server
- 

$\tau_s$  time units for the weakest worker to compute its gradient, HGO only waits for  $\tau < \tau_s$  time units at each iteration, and the workers adjust to this time budget. However, note that the procedure of identifying the adequate block of coordinates and the workers that computed each of these coordinates (steps 8, 9, 10 and 11 in Algorithm 1) induces an overhead in computation when considering the total time complexity of the algorithm. Overall, the relative speed of HGO compared to SGD depends on the relation between  $\tau - \tau_s$  and the overhead cost, which can be compared to  $\mathcal{O}(q \log(q) \frac{\bar{\lambda} \tau}{\bar{s}})$ , where  $\bar{\lambda}$  is the average computation rate and  $\bar{s}$  is the average mini-batch size of the workers.

## V. THEORETICAL GUARANTEES

Inspired by prior works in the homogeneous setting [8], [9], [11], [38], we show that our algorithm eventually reaches a ball centered at a local optimum, whose radius is a function of the statistical error. More formally, we show that, when

$K \rightarrow \infty$ , HGO eventually outputs a parameter estimate  $w$  such that

$$\mathbb{E} \|\nabla \mathcal{L}(w)\|^2 \leq \frac{d \Delta U}{b(1 - \Delta V)}$$

where  $b = \liminf_{k \geq 1} \mathbb{E}[\|\mathcal{B}_k\|]$ ,  $U = \max_{w \in \mathbb{R}^d} U_w$ ,  $V = \max_{w \in \mathbb{R}^d} V_w$

$$U_w = \frac{1}{|C|} \sum_{j=1}^d \sum_{i \in C} \left( \frac{M_w}{s'_i} + \delta_{i,j}^2 \right), \quad V_w = \frac{1}{|C|} \sqrt{\sum_{j=1}^d \left( \sum_{i \in C} \frac{Q_w}{s'_i} \right)^2}$$

with  $s'_i = \frac{s_i(|\mathcal{S}_i|-1)}{|\mathcal{S}_i|-s_i}$  and  $\Delta = \Delta(v, f)$ .

Figure 1 illustrates the trajectory of the algorithm in the context where there exists a unique optimal solution  $w^*$ . In Theorem 1 and 2, we present the formal convergence result of our algorithm. In doing so, we divide the parameter space into two regions  $\mathcal{W}$  and  $\mathbb{R}^d \setminus \mathcal{W}$ , with

$$\mathcal{W} = \left\{ w \mid \min_{\mathcal{B} \subseteq [d]} \left\{ (1 - \Delta V_w) \|\nabla \mathcal{L}(w)\|_{\mathcal{B}}^2 - \Delta U_w \right\} > 0 \right\}.$$

We denote by  $[\nabla \mathcal{L}(w)]_{\mathcal{B}}$  the partial gradient restricted to the coordinates in  $\mathcal{B}$ . Specifically, for all  $u = (\{u\}_1, \dots, \{u\}_d) \in \mathbb{R}^d$ ,  $u' = [u]_{\mathcal{B}} \in \mathbb{R}^d$  is such that  $\{u'\}_j = \{u\}_j$  if  $j \in \mathcal{B}$ , and 0 otherwise.

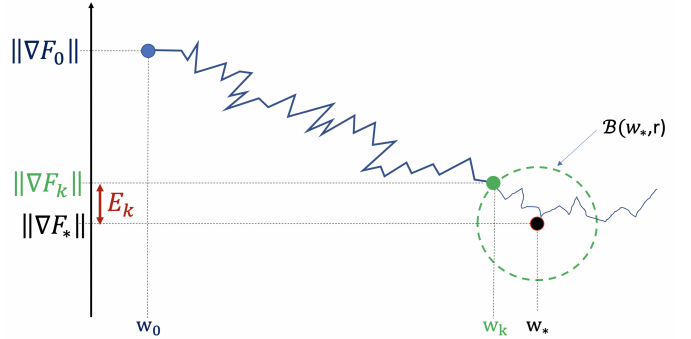


Fig. 1: As long as the parameter vector  $w_k$  is inside  $\mathcal{W}$ , which means that the gradient norm is still big enough, the algorithm will keep approaching the ball  $\mathcal{B}(w_*, r)$  centered at a local optimum, with a radius  $r$  (which is a function of the statistical error). When the condition is no longer satisfied, the perturbations induced by the presence of Byzantine workers will prevent progress towards the optimum. Then, the smaller  $r$ , the better the convergence guarantee of the algorithm in the presence of Byzantine faults.

The condition  $(1 - \Delta V_w) \|\nabla \mathcal{L}(w)\|_{\mathcal{B}}^2 > \Delta U_w$  can be seen as a generalization of the *variance-norm* condition first introduced in [8]. This work was considering the special case of SGD ( $b = d$  and  $\mathcal{B} = [d]$ ) with unbiased estimation ( $\forall (i, j) \in [N] \times [d], \delta_{i,j} = 0$ ), uniform upper bound on the variance ( $Q_w = 0$ ), and a fixed mini-batch of size 1 ( $\forall i \in [n], s_i = 1$ ). In this special case, our condition reduces to the variance-norm condition, presented in Proposition 1 of [8]. The authors also claimed that their condition will be easier to satisfy when using larger mini-batch sizes. Our

formulation confirms this claim: increasing the mini-batch size ( $s_i$ ) increases the value of the right-hand side of the condition inequality and decreases the left-hand side one, making it easier to achieve a satisfying outcome in many problems.

#### A. Convergence analysis

Before presenting our main convergence results, note that, given a parameter vector  $w_k$ , it is hard to demonstrate progress towards a local optimum without first demonstrating that  $\text{AG}_{w_k}$  is pointing in the same direction as the actual gradient of the loss function. Specifically, the scalar product between the aggregated gradient  $\text{AG}_{w_k}$  and  $\nabla \mathcal{L}(w_k)$  must be strictly positive. We show below in Lemma 1 that this condition holds true in our setting whenever  $w_k \in \mathcal{W}$  and the aggregation rule  $\text{AR}$  we use is  $\Delta(v, f)$ -robust.

**Lemma 1.** *Suppose that Assumptions 3 and 4 hold true. Consider the  $k$ -th step of Algorithm 1. If  $\text{AR}$  is  $\Delta(v, f)$ -robust at  $w_k$  and  $w_k \in \mathcal{W}$ , then, denoting  $\Delta = \Delta(v, f)$ , we have:*

$$\mathbb{E} [\langle \text{AG}_{w_k}, [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle] > \frac{(1 - \Delta V_{w_k}) \|\nabla \mathcal{L}(w_k)\|_{\mathcal{B}_k}^2 - \Delta U_{w_k}}{2}$$

Using this result, we obtain the convergence properties of HGO for strongly convex and non-convex functions. Under Assumption 1, we define  $L = \max_{i \in [d]} L_i$ .

**Theorem 1** (Strongly convex convergence). *Suppose that Assumptions 1, 2, 3 and 4 hold true. Consider Algorithm 1 with a constant learning rate  $\gamma$ . If  $\gamma < \frac{1}{\sqrt{dL}}$  and  $\Delta V < 1$ , then*

$$\mathbb{E} [\mathcal{L}(w_K) - \mathcal{L}^*] \leq \left(1 - \frac{b}{d} \mu \gamma (1 - \Delta V)\right)^{K-1} (\mathcal{L}(w_1) - \mathcal{L}^* - H) + H$$

where  $\mathcal{L}^*$  denotes the minimum value of  $\mathcal{L}$  on  $\mathbb{R}^d$ ,  $b = \min_{k \in [K]} \mathbb{E}_{\mathcal{B}_k} [|\mathcal{B}_k|]$ ,  $U = \max_{k \in [K]} U_{w_k}$ ,  $V = \max_{k \in [K]} V_{w_k}$ , and  $H = \frac{d \Delta U}{2b\mu(1-\Delta V)}$ .

Unlike convex functions, non-convex functions may have multiple local minima, making convergence analysis more complex [38], especially for coordinate descent methods [17], [40], which are closer to our algorithm. While it is hard in general to upper-bound the optimality gap as in previous theorem, we upper-bound the average (and thus the minimum) gradient norm of the cost function, evaluated using the sequence of parameter vectors  $\{w_k\}_{k \in [K]}$  generated through the iterations  $1, \dots, K$ .

**Theorem 2** (Non-convex convergence). *Suppose that Assumptions 1, 3 and 4 hold. If  $\gamma < \frac{1}{\sqrt{dL}}$  and  $\Delta V < 1$  then*

$$\min_{k \in [K]} \mathbb{E} [\|\nabla \mathcal{L}(w_k)\|^2] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{Kb\gamma(1 - \Delta V)} + \frac{d \Delta U}{b(1 - \Delta(v, f)V)}$$

where  $U = \max_{k \in [K]} U_{w_k}$ ,  $V = \max_{k \in [K]} V_{w_k}$  and  $b = \min_{k \in [K]} \mathbb{E}_{\mathcal{B}_k} [|\mathcal{B}_k|]$ .

In Appendix A, we provide some guidelines on how to choose the learning rate  $\gamma$ . We also show through examples how our work can be considered as a generalization of previous results in the field of Byzantine resilience and gradient-based optimization algorithms.

#### B. Trade-offs

Our theoretical results highlight some interplay between the convergence speed, the computation power of the workers, as well as the dataset partitioning and the presence of Byzantine workers. We discuss some of them below.

**Convergence vs time complexity.** The robustness coefficient  $\Delta(v, f)$  is a key factor of the aggregation error. Its value depends on the quality of the aggregation rule we use, but also on the number of Byzantine workers within the selection of random workers. For a fixed number  $f$  of Byzantine devices, increasing the number of devices sampled at every round  $q$  also increases the number of correct workers amongst the quorum of size  $v$  we use for the aggregation. This has a direct impact on the convergence rate as well as on the statistical error. As a matter of fact, one can see in both theorems that the first term of the right-hand side inequality decreases when  $\Delta(v, f)$  decreases. Recall from Section III-C that  $\Delta(v, f)$  is decreasing with respect to  $v$ . This simply means that a bigger sample size  $q$  implies a faster convergence rate. The statistical error is also positively correlated to  $\Delta(v, f)$ ; hence, also to  $q$ . However, note that, as explained in Section IV-B, the time complexity of the algorithm also increases with  $q$ . In short, a better theoretical convergence implies a bigger time complexity.

**Heterogeneity vs convergence.** From our theoretical results, we can observe that increasing the mini-batch size  $s_i$  of the workers improves the statistical error as well as the convergence rate. On the other hand, these two convergence properties are also linked to the size  $|\mathcal{B}_k|$  of the block  $\mathcal{B}_k$  constructed by the server at iteration  $k$ . In fact, increasing  $b = \min_{k \in [K]} \mathbb{E}_{\mathcal{B}_k} [|\mathcal{B}_k|]$  also increases the convergence rate and reduces the statistical error. To increase  $b$ , we can either:

- 1) Decrease the minimum number of workers  $v$  to execute the aggregation. However, this quantity is lower-bounded by a function of the ratio of Byzantine workers the algorithm can tolerate; hence, it cannot be decreased arbitrarily without impeding the Byzantine resilience of the algorithm.
- 2) Decrease the mini-batch sizes, so that every worker dedicates its resources to computing more blocks ( $\lambda_i \tau = b_i s_i$ ). However, note that it would induce a new trade-off, as diminishing the mini-batch sizes has a negative effect on the convergence of the algorithm.
- 3) The server can try to re-weight the worker sampling to contact the powerful workers more often (the ones with large computational capabilities  $\lambda_i$ ), hence mechanically increasing the block size and the mini-batch size together. This would improve the convergence rate, but may exclude workers with valuable data from the learning procedure, which certainly leads to a worse statistical error when data are heterogeneous.

**Byzantine tolerance vs hardware heterogeneity.** Maybe the most interesting takeaway we can extract from our results is the interplay between hardware heterogeneity and Byzantine tolerance of the algorithm. In both results, we introduce a

sufficient condition for convergence, namely:  $\Delta(v, f)V < 1$ . This inequality formalizes the interplay between the Byzantine tolerance (through the robustness coefficient  $\Delta(v, f)$ ) and the computational power of the workers (captured in the variable  $V$ ). To tolerate a bigger Byzantine impact,  $V$  must decrease in order to follow a gradient step at round  $k$ . Decreasing  $V$  implies to increase the size of the mini-batches selected by the workers, to reduce the variance. This can be achieved by instructing the workers to favor the mini-batch size over the number of coordinates, when allocating the available computation resources. However, when the computational capabilities are not high enough, decreasing the number of coordinates leads to a small block size  $b$ , which results in poor convergence properties. Conversely, if each worker has strong capabilities, it will be possible to increase both  $b$  and  $s_i$ , in order to satisfy the condition and improve the convergence properties at the same time. In other words, the more weak devices are present in the network, the less the algorithm can tolerate Byzantine behaviors.

## VI. EMPIRICAL EVALUATION

To demonstrate the practical relevance of our optimization scheme in a Byzantine-prone environment with heterogeneous participants, we ran an exhaustive set of experiments to evaluate the performance of HGO on several classification tasks. The main objective of this section is to demonstrate that our protocol improves the participation of slow devices, and outperforms traditional schemes such as SGD (in terms of final accuracy of the model); hence highlighting the importance of our method when learning with heterogeneous and Byzantine participants. To do so, we first present a set of experiments on simple models such as multinomial logistic regression. Then, we propose an extension of HGO to more complicated models, such as feed-forward neural networks, and discuss some future direction for our protocol in the context of convolutional neural networks. To simplify the presentation, we evaluate the performance of HGO (used with a vanilla SGD update rule) against vanilla SGD to eliminate the impact of more advanced solvers such as NAG [22], RMSprop [23], AdaGrad [24] or Adam [25], that are still misunderstood from a technical viewpoint.

Due to page limitation, we only present a few experiments that support our theoretical analysis. A more complete set of experiments, involving more ML models and datasets, is available in Appendix C. An Android implementation of our algorithm is also available to execute it on real smartphones. The source code of our experiments and the Android implementation are available in the following GitHub repositories, respectively: <https://github.com/karimboubouh/HgO> and <https://github.com/karimboubouh/HgOApp>.

### A. Experimental setup

**Models and datasets.** We evaluated our procedure on a broad class of models, including linear regression, binary logistic regression and support vector machine, on several

benchmark datasets (such as Boston<sup>2</sup>, Wisconsin Breast Cancer<sup>3</sup>, or Phishing<sup>4</sup>) to confirm our theoretical results. But we also considered more complex setups, namely: multinomial logistic regression and feed-forward neural network trained on MNIST<sup>5</sup> and Fashion-MNIST<sup>6</sup>. In the main paper, we only present the last two setups, but our findings were consistent across the set of experiments we considered.

Note that, for a fair comparison, we use the same learning rate and mini-batch size for SGD and HGO in every experiments.

**Network architecture and hardware specification.** We implement the parameter-server architecture where a trusted server communicates through message passing with a set of workers, to learn the model. To distinguish the hardware specifications of workers, we classify them in three categories characterizing their computational capabilities, namely: “weak”, “average” and “powerful” (see Appendix C3 for more details). Each category represents a set of workers whose computation rates are normally distributed over a fixed mean value that depends on the learning task. We consider several computation profiles for the network:

- $C_{standard}$ : 30% of weak devices, 40% average devices and 30% powerful devices;
- $C_{weak}$ : all devices are weak;
- $C_1$ : 90% of weak and 10% of average devices;
- $C_2$ : 70% of weak and 30% of average devices;
- $C_{powerful}$ : all devices are powerful.

When the computation profile of the network is not specified, we assume that we are in the standard scenario  $C_{standard}$ .

At each iteration, the server only contacts a random set of  $q$  workers, and aggregates their responses using a given aggregation rule, after a waiting time  $\tau$ . By default, we set  $\frac{q}{n} = 0.8$ , and use *Average* and *Median* (a special case of Trimmed Mean) as aggregation rules in honest and Byzantine environments, respectively.

**Data heterogeneity.** We experiment both homogeneous and heterogeneous partitioning of datasets over the workers. In the homogeneous case, each worker is assigned a non-overlapping independent and identically distributed sample from the training set. We construct heterogeneous versions of the datasets by restricting the classes detained by each workers. Specifically, in our experiments, each worker only has access to data points for two classes out of ten from the MNIST and Fashion-MNIST datasets.

**Byzantine workers.** We use two state-of-the-art Byzantine attacks [41], [42] to evaluate the impact of our new learning scheme on the robustness of existing aggregation rules (Krum [8], Trimmed Mean [37] and Aksel [9]). These attacks exploit the normal distribution of correct estimates to construct Byzantine values that are still within a few standard deviations

<sup>2</sup><http://lib.stat.cmu.edu/datasets/boston>

<sup>3</sup>[https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+\(diagnostic\)](https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic))

<sup>4</sup><https://archive.ics.uci.edu/ml/datasets/phishing+websites>

<sup>5</sup><http://yann.lecun.com/exdb/mnist/>

<sup>6</sup><https://github.com/zalandoresearch/fashion-mnist>

from the true mean. Byzantine workers start implementing the aforementioned attacks at iteration 10 of each experiment. For comparison, we always plot the test accuracy of averaging under no attack.

### B. Impact of waiting time on accuracy and runtime

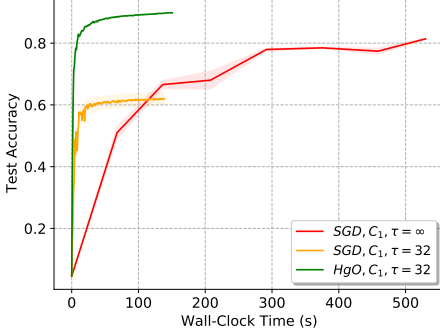


Fig. 2: Training a multinomial logistic regression on MNIST, heterogeneously partitioned over  $n = 100$  honest workers. The server averages the estimates of  $q = 80$  random workers at each iteration. When  $\tau = \infty$ , the server waits for the slowest device to send its message. When  $\tau = 32$ , the server waits for 32 time units and only aggregates the received messages.

In Figure 2, we show the evolution of the accuracy of a multinomial regression on MNIST trained with SGD and HGO.<sup>7</sup> When we do not impose any limit on the waiting time of the server ( $\tau = \infty$ ), it must wait for every worker to complete its computing task before executing any aggregation. Consequently, the speed of SGD is limited by the hardware specifications of the weakest device of the network. While this approach is indeed slow, it allows the model to learn from all workers’ local datasets, which results in a good accuracy. A practitioner might be tempted to reduce the waiting time by fixing a time limit for each iteration (e.g.  $\tau = 32$  time units). By doing so, we can observe a substantial speedup, because the server discards the workers that cannot provide a response within the time limit. However, when the dataset is not homogeneously partitioned, the model learned with SGD will miss the data points held by slow workers, resulting in a poor final accuracy.

With HGO, even when limiting the server waiting time, the workers can adjust their computing task to the time budget, by only computing a few partial derivatives using a few data points in the mini-batch. That way, the model is visiting the full dataset either through full gradients (computed by powerful workers) or partial derivatives (computed by weaker workers). This results in a much better overall accuracy of the model, without slowing down the overall execution of the procedure. In the next experiments, we always set the waiting time  $\tau$  to 32 for HGO and SGD.

<sup>7</sup>To measure this runtime, we use the actual wall-clock time, which gives a clear idea of the runtime, as opposed to the number of algorithm steps, which simply gives an idea of the convergence rate of the protocols.

### C. Impact of heterogeneous partitioning

As we explained above, the principal advantage of HGO is to allow the server to receive information from *all* the workers at each iteration. Note that, when the dataset is homogeneously partitioned over the workers, it is sufficient to have a few workers in order to get a representative sampling of the full dataset. This situation is illustrated in Figure 3a, where SGD reaches a good accuracy, even when not all workers are participating in the training process (except for the weakest scenario  $C_{weak}$ , where no worker is able to compute a full gradient). However, in the heterogeneous case illustrated in Figure 3b, HGO clearly outperforms SGD, as discarding weaker workers now comes with the drawback of losing valuable information from their local datasets. While SGD’s performance is strongly correlated with the computation capabilities of the workers, HGO shows roughly the same convergence properties, independently of the hardware profile of the workers. Interestingly, using HGO allows to reach top performance, even with a network full of weak devices, which is impossible with standard learning schemes.

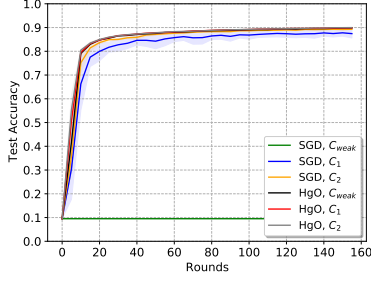
### D. Improving weak networks with a few average devices

In the previous experiments (Figure 3), one can see that HGO shows nearly the same convergence pattern for all scenarios. In the experiment reported in Figure 4, we voluntarily set the computation rates of the weak devices to very low values, in order to get a lower accuracy using the weakest scenario  $C_{weak}$ . While using HGO with a network full of weak workers still shows good performance, we demonstrate that replacing only 10% of these weak devices by average workers is sufficient to match the convergence quality of an all-powerful network on MNIST. This experiment is very interesting, and motivates the use of HGO in industry-scale machine learning applications. For instance, a company may train a complex machine learning model using HGO with data available on smartphones, which can be considered as weak devices. Then, it can introduce a few powerful computers to boost the learning performance. The result would be similar to a scenario where all the data is relocated into one location, and trained with strong computers using SGD.

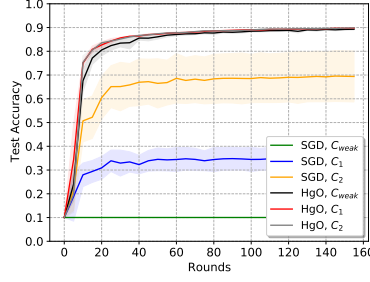
### E. Conditioning Byzantine resilience to computational heterogeneity

Another interesting takeaway of our analysis is the interplay between the strength of Byzantine workers and the computational heterogeneity, represented by the condition  $\Delta(v, f)V < 1$ . This condition has a direct impact on the convergence rate, as well as on the statistical error of the algorithm. In fact, one can act on the robustness coefficient  $\Delta$  by varying the aggregation rules, the number of Byzantine workers or the Byzantine attack strategy implemented in a given execution. One can also act on the variable  $V$  by changing the data partitioning as well as the network computation profile. In the three experiments in Figure 5, we illustrate these concepts, and also implement scenarios where we intentionally dissatisfy the condition  $\Delta V < 1$ , for instance, by choosing a weaker

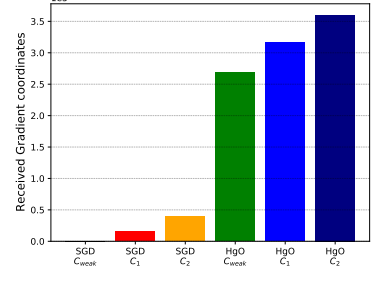




(a) Homogeneous dataset partitioning



(b) Heterogeneous dataset partitioning



(c) Heterogeneous dataset partitioning

Fig. 3: Training a multinomial logistic regression on MNIST, using  $n = 100$  workers in an honest setting. HGO shows improved convergence properties (final accuracy, convergence rate) w.r.t. SGD, even in the weakest scenario  $C_{weak}$  where all workers have low hardware profiles. In particular, the gap widens when the dataset is heterogeneously partitioned (see Figure 3b). Figure 3c demonstrate the fact that HGO receives more estimations than SGD, even in very restricted network profiles ( $C_{weak}$ : all the workers are weak devices), where no worker is able to compute a descent estimation in the case of SGD.

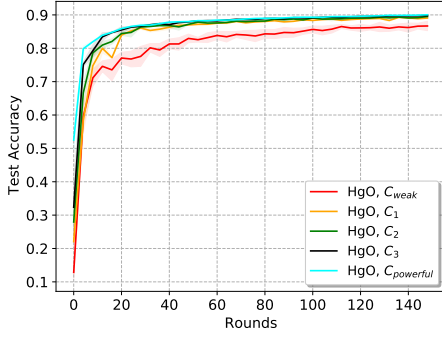


Fig. 4: Training a multinomial logistic regression on MNIST, homogeneously partitioned over  $n = 100$  honest workers. The server averages the estimates of  $q = 80$  random workers at each iteration. Interestingly, replacing 10% of weak devices by average ones ( $C_1$ ) is sufficient to match the performance of ( $C_{powerful}$ ).

aggregation rule (Krum) in 5a, by implementing a strong attack strategy (FOE) in 5b, or by increasing the number of Byzantine workers ( $f = 20, 30$ ) in 5c.

#### F. Extension to neural networks

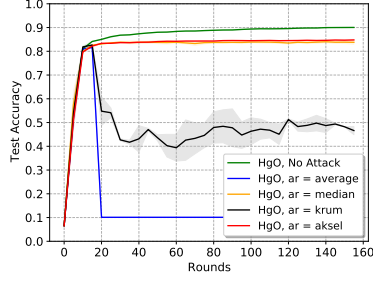
Note that simple linear models have separable cost functions, which means that computing partial derivatives can be done independently. However, for more complex models such as neural networks, computing the gradient relies on the back-propagation algorithm and the chain rule. This makes the computation of a partial derivative much more dependent on the model architecture than in the linear case. Most importantly, picking a block of random coordinates from the full gradient will result in an unbalanced updates between the network layers. In this section, we present a variant of HGO, where workers only update sub-network at each round of the learning.

**Adaptation of HGO to Feed-forward networks.** Let us consider a feed-forward network with  $M$  layers. To compute a partial gradient at each round of the protocol, each worker

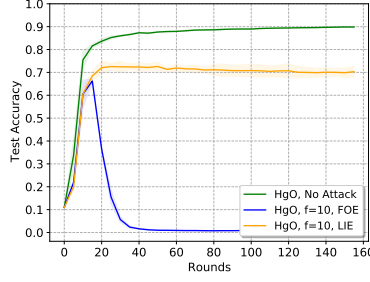
samples a subset of the network weights' on the  $M - 1$  first layers of the network, hence building a sub-network. The output layer should be kept unchanged for consistency purposes (the model should still be able to classify within the same number of classes). Naturally, the number of sampled weights in each layer depends upon the computational capabilities of the worker. Then, each worker computes their gradient on the obtained sub-network to mimic the computation of a partial gradient at each round. This procedure is illustrated in Figure 6 for a simple 3-layers feed-forward network.

The gradient of the sub-network (whose dimension is less than the original gradient) can be considered as a partial gradient, or a block of random gradient coordinates, covering all the layers of the neural network. For simpler models, the random block is a vector and the block size is an integer. However, for feed-forward neural networks, a block is a tensor, and its size is a vector of integers. For example, for a feed-forward neural network with input size 784, one hidden layer with 30 neurons and an output layer with 10 classes, a block size will be written in the following format:  $[98, 20, 10]$ . This means that 98 out of 784 weights in the first layer, and 20 out of 30 weights in the second layer, will be sampled and updated at each round (as explained above, the last layer remains unchanged). As illustrated in Figure 7, this algorithm extension produces results similar to multinomial logistic regression, therefore demonstrating the practical relevance of HGO in the case of neural networks.

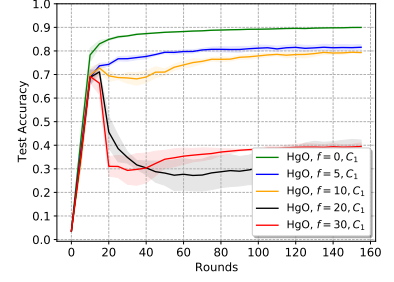
**Relation to dropout.** Note that this alternative approach to HGO can be linked to the dropout technique [43], widely used among ML practitioners. When employing dropout in a neural network with classical learning schemes, a fraction of weights are ignored in the training phase, with some probability. In HGO, each worker will dropout a fraction of the weights, depending on its computing capabilities, in a deterministic fashion. Powerful devices can use the full neural network without applying the dropout. Like regularization techniques, the dropout approach prevents over-fitting, and this might be another explanation of the better final test accuracy of HGO



(a) Homogeneous dataset partitioning



(b) Heterogeneous dataset partitioning



(c) Heterogeneous dataset partitioning

Fig. 5: Training a multinomial logistic regression on MNIST, with a total of  $n = 100$  workers. The server aggregates the partial gradients of  $q = 80$  random workers at each iteration. By default, we use Median as an aggregation rule and set  $f = 10$ . In each experiment, we vary the number of Byzantine workers, their attack and the aggregation rule: all these factors impact the variable  $\Delta$ , leading to different convergence properties. In the three experiments, we implement scenarios to intentionally dissatisfy the condition  $\Delta V < 1$ , by working on the robustness coefficient  $\Delta$ : for instance, by choosing a weaker aggregation rule (Krum) in (a), by implementing a strong attack strategy (FOE) in (b), and by increasing the number of Byzantine workers ( $f = 20, 30$ ) in (c).

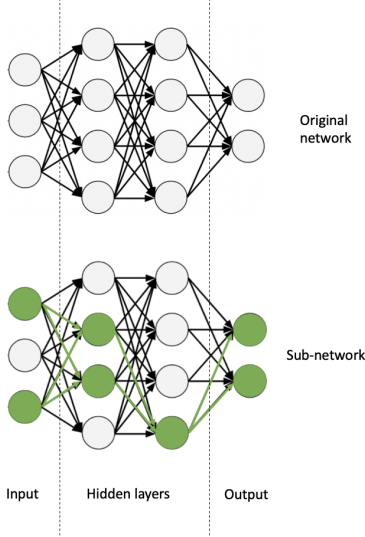


Fig. 6: Constructing a sub-network from a fully connected feed-forward neural network.

over SGD, besides the fact that HGO includes all local datasets in the training.

**Advanced architectures.** More advanced architectures like convolutional neural networks (CNN) need a different way of constructing a sub-network. In fact, the classification phase of a CNN is a simple feed-forward network, on which we can easily apply the sub-network technique. However, the feature extraction phase (convolutions and pooling) is challenging. In this work, we limit our extension to feed-forward neural networks, and leave the CNN extension for future work.

## VII. CONCLUSION

We presented HGO, a Byzantine-resilient distributed learning scheme whose iteration cost can be adapted to the capabilities of workers (e.g., smartphones). HGO encourages

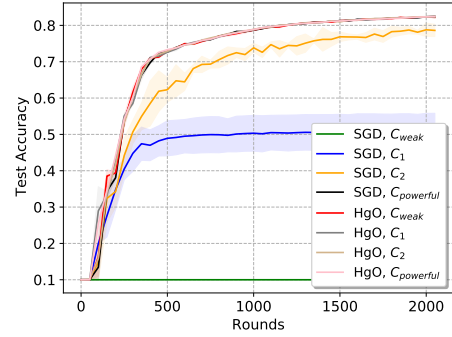


Fig. 7: Training a feed-forward neural network (fully connected network with 784 inputs, one hidden layer of 128 neurons and 10 outputs) on a heterogeneously partitioned Fashion-MNIST dataset, with a total of  $n = 100$  honest workers. Depending on its hardware capability, each device uses a random sub-network to compute the gradient estimate. Powerful devices use the full neural network.

the participation of slower devices, improving the accuracy of the model when the participants do not share the same dataset. Our protocol also tolerates Byzantine devices, by combining a randomized sampling of devices with a robust aggregation scheme. We proved HGO's convergence under a non-trivial combination of hypotheses, namely: Byzantine failures, computational heterogeneity and data heterogeneity. Our theory demonstrated important tensions between relevant quantities such as the statistical error rate, the impact of Byzantine workers, the level of inaccuracy of honest workers, the time complexity of the algorithm and its convergence rate, which we also supported empirically.

HGO clearly outperforms SGD in the case of non-identically distributed datasets by aggregating the estimates of all workers, regardless of their computational capabilities. This contrasts with SGD, which imposes the same workload on all workers, and therefore discards some workers (which data may be very valuable to the training). On a more ethical side,



our work encourages data privacy by making smartphones (which are the most important personal data carrier nowadays) contribute directly to AI projects, without exposing local data to third parties.

Furthermore, this work opens an interesting line of open questions, among which: 1) Can this work be extended to second-order optimization and to convolutional neural networks? 2) Which computation power profile is needed to reach a target accuracy within a given time budget? And finally, 3) what is the impact of network bandwidth on the convergence properties? We leave these questions open for future works.

## REFERENCES

- [1] M. Abadi *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.
- [2] J. Konečný, H. McMahan, D. Ramage, and P. Richtárik, “Federated optimization: Distributed machine learning for on-device intelligence,” *ArXiv*, vol. abs/1610.02527, 2016.
- [3] L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, Jul. 1982. [Online]. Available: <https://doi.org/10.1145/357172.357176>
- [4] R. M. Gower, N. Loizou, X. Qian, A. Sailanbayev, E. Shulgin, and P. Richtárik, “SGD: General analysis and improved rates,” in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 5200–5209. [Online]. Available: <https://proceedings.mlr.press/v97/qian19b.html>
- [5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: <https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf>
- [6] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar, “An asynchronous parallel stochastic coordinate descent algorithm,” *J. Mach. Learn. Res.*, vol. 16, no. 1, p. 285–322, jan 2015.
- [7] N. A. Lynch, *Distributed algorithms*. Elsevier, 1996.
- [8] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent,” ser. NeurIPS’17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 118–128.
- [9] A. Boussetta, E.-M. El-Mhamdi, R. Guerraoui, A. Maurer, and S. Rouault, “AKSEL: Fast Byzantine SGD,” in *24th International Conference on Principles of Distributed Systems (OPDIS 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), Q. Bramer, R. Oshman, and P. Romano, Eds., vol. 184. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021, pp. 8:1–8:16. [Online]. Available: <https://drops.dagstuhl.de/opus/volltexte/2021/13493>
- [10] G. Damaskinos, E. M. El Mhamdi, R. Guerraoui, R. Patra, and M. Taziki, “Asynchronous Byzantine machine learning (the case of SGD),” in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1145–1154. [Online]. Available: <http://proceedings.mlr.press/v80/damaskinos18a.html>
- [11] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, “Byzantine-robust distributed learning: Towards optimal statistical rates,” 2018.
- [12] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, “Draco: Byzantine-resilient distributed training via redundant gradients,” 2018.
- [13] D. Alistarh, Z. Allen-Zhu, and J. Li, “Byzantine stochastic gradient descent,” in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018, pp. 4613–4623. [Online]. Available: <https://proceedings.neurips.cc/paper/2018/file/a07c2f3b3b907aaf8436a26c6d77f0a2-Paper.pdf>
- [14] D. Data and S. Diggavi, “Byzantine-tolerant distributed coordinate descent,” in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 2724–2728.
- [15] Z. Yang and W. U. Bajwa, “Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning,” *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 4, pp. 611–627, 2019.
- [16] E.-M. El-Mhamdi, R. Guerraoui, A. Guirguis, L. N. Hoang, and S. Rouault, “Genuinely distributed byzantine machine learning,” New York, NY, USA: Association for Computing Machinery, 2020, p. 355–364. [Online]. Available: <https://doi.org/10.1145/3382734.3405695>
- [17] S. J. Wright, “Coordinate descent algorithms,” *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, 2015.
- [18] Q. Tao, K. Kong, D. Chu, and G. Wu, “Stochastic coordinate descent methods for regularized smooth and nonsmooth losses,” in *Machine Learning and Knowledge Discovery in Databases*, P. A. Flach, T. De Bie, and N. Cristianini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 537–552.
- [19] P. Richtárik and M. Takáč, “Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function,” *Math. Program.*, vol. 144, no. 1–2, p. 1–38, apr 2014. [Online]. Available: <https://doi.org/10.1007/s10107-012-0614-z>
- [20] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, 1st ed. Springer Publishing Company, Incorporated, 2014.
- [21] C. Xie, O. Koyejo, and I. Gupta, “Generalized byzantine-tolerant sgd,” 2018.
- [22] Y. Nesterov, “A method for solving the convex programming problem with convergence rate  $o(1/k^2)$ ,” *Proceedings of the USSR Academy of Sciences*, vol. 269, pp. 543–547, 1983.
- [23] G. Hinton, N. Srivastava, and K. Swersky, “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent,” *Cited on*, vol. 14, no. 8, p. 2, 2012.
- [24] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011. [Online]. Available: <http://jmlr.org/papers/v12/duchi11a.html>
- [25] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2015.
- [26] A. Khaled, K. Mishchenko, and P. Richtárik, “Tighter theory for local sgd on identical and heterogeneous data,” in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 4519–4529. [Online]. Available: <https://proceedings.mlr.press/v108/bayoumi20a.html>
- [27] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. U. Stich, “A unified theory of decentralized sgd with changing topology and local updates,” in *ICML*, 2020, pp. 5381–5393. [Online]. Available: <http://proceedings.mlr.press/v119/koloskova20a.html>
- [28] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, “On the convergence of fedavg on non-*id* data,” in *International Conference on Learning Representations*, 2020. [Online]. Available: <https://openreview.net/forum?id=HJxNANvtdS>
- [29] D. Data, L. Song, and S. Diggavi, “Data encoding methods for byzantine-resilient distributed optimization,” in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 2719–2723.
- [30] S. Rajput, H. Wang, Z. B. Charles, and D. Papailiopoulos, “Detox: A redundancy-based framework for faster and more robust gradient aggregation,” in *NeurIPS*, 2019.
- [31] X. Lian, Y. Huang, Y. Li, and J. Liu, “Asynchronous parallel stochastic gradient for non convex optimization,” in *NIPS*, pages 2737–2745, 2015.
- [32] J. Jiang, B. Cui, C. Zhang, and L. Yu, “Heterogeneity-aware distributed parameter servers,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 463–478. [Online]. Available: <https://doi.org/10.1145/3035918.3035933>
- [33] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, “signsgd: Compressed optimisation for non-convex problems,” 2018.
- [34] T. Vogels, S. P. Karimireddy, and M. Jaggi, “Powersgd: Practical low-rank gradient compression for distributed optimization,” in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox,

- and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: <https://proceedings.neurips.cc/paper/2019/file/d9fbed9da256e344c1fa46bb46c34c5f-Paper.pdf>
- [35] H. Sun, Y. Shao, J. Jiang, B. Cui, K. Lei, Y. Xu, and J. Wang, “Sparse gradient compression for distributed sgd,” in *Database Systems for Advanced Applications*, G. Li, J. Yang, J. Gama, J. Natwichai, and Y. Tong, Eds. Cham: Springer International Publishing, 2019, pp. 139–155.
  - [36] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, “Deep gradient compression: Reducing the communication bandwidth for distributed training,” in *International Conference on Learning Representations*, 2018. [Online]. Available: <https://openreview.net/forum?id=SkhQHMW0W>
  - [37] C. Xie, O. Koyejo, and I. Gupta, “Phocas: dimensional byzantine-resilient stochastic gradient descent,” 2018.
  - [38] L. Bottou, F. E. Curtis, and J. Nocedal, “Optimization methods for large-scale machine learning,” *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.
  - [39] L. Nguyen, P. H. Nguyen, M. van Dijk, P. Richtarik, K. Scheinberg, and M. Takac, “SGD and hogwild! Convergence without the bounded gradients assumption,” ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 3750–3758. [Online]. Available: <http://proceedings.mlr.press/v80/nguyen18c.html>
  - [40] Y. Nesterov, “Efficiency of coordinate descent methods on huge-scale optimization problems,” *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341–362, 2012.
  - [41] C. Xie, S. Koyejo, and I. Gupta, “Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation,” 2019.
  - [42] G. Baruch, M. Baruch, and Y. Goldberg, “A little is enough: Circumventing defenses for distributed learning,” in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019, pp. 8635–8645. [Online]. Available: <https://proceedings.neurips.cc/paper/2019/file/ec1c59141046cd1866bbcbdfb6ae31d4-Paper.pdf>
  - [43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: <http://jmlr.org/papers/v15/srivastava14a.html>
  - [44] A. Mood, *Introduction to the Theory of Statistics*. McGraw-Hill, 1950.