DEMOCRATIZING MACHINE LEARNING

RESILIENT DISTRIBUTED LEARNING WITH HETEROGENEOUS PARTICIPANTS

Anonymous authors
Anonymous affiliation

October 16, 2021

ABSTRACT

Classical algorithms in distributed learning usually impose a uniform workload on all participating devices, typically discarding slower workers in each iteration of the learning process. This not only induces a sub-optimal utilization of the available computational resources, but also a poor learning accuracy, as critical data only held by slower devices are discounted from the learning. We propose HGO, a distributed learning algorithm with parameterizable iteration costs that can be adjusted to the computational capabilities of different devices, thereby encouraging the participation of slower devices. Besides being adaptive, HGO is also robust in the sense that it tolerates faulty devices, by combining a randomized sampling of devices with a robust aggregation scheme. We convey the convergence of HGO for both non-convex and strongly convex settings, without assuming any specific partitioning of the data over the devices. By doing so, we characterize, for the first time, the interplay between the statistical error, the convergence rate, the Byzantine resilience, the per-iteration cost and the run-time complexity of distributed learning. Particularly, we capture the trade-off between the impact of Byzantine workers (which is a function of the mini-batch size, as we show) and the computation rate of workers (which is critical for convergence). Empirically, we evaluate HGO in a distributed setting, for a panoply of models and datasets.

1 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.[1] Each day, these devices generate a massive amount of data that must be processed locally (on the device that generated it), both for infrastructure cost efficiency and privacy reasons. Distributed machine learning (ML) proposes to implement this requirement as follows: several devices collaboratively learn a common model by exchanging parameters, possibly through a server machine [1, 16]. However, standard solutions to distributed ML do not provide an obvious way to account for heterogeneity between devices, neither in terms of data generation nor in terms of hardware.

While classical optimization algorithms like *Stochastic Gradient Descent* (SGD) have proved their utility amongst practitioners, they typically assume that participating devices have similar hardware specifications. While this requirement might be satisfied in data centers equipped with powerful co-located computers steadily connected to the power grid and to the network, personal computing devices owned by general public (such as smartphones or laptops) often behave in an unpredictable way. For example, they may often disconnect from the network, become frequently inactive to save battery consumption, or crash accidentally. Classical algorithms are not well suited for such handheld devices, as they notoriously discard weaker devices (with low computational powers) from the learning process. This leads to (1) sub-optimal usage of available computing resources and (2) poorer learning accuracy, as data held by weaker devices is excluded from the learning process. On top of that, some devices might get hacked or manipulated by malicious parties, and inject arbitrary perturbations to the system. Such devices are technically referred to as *Byzantine* [17].

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

In particular, we recognize three critical issues when deploying classical distributed ML algorithms on handheld devices, namely: *computational heterogeneity*, *data heterogeneity*, and *Byzantine failures*. These issues have received a lot of attention so far, but prior works either consider them individually (e.g. [4, 6, 11, 2, 26, 27, 19, 7, 29, 14, 15, 30, 18, 13]), or consider only two of these issues at a time (e.g. [10, 24, 12, 8]).[2] We propose HGO,[3] a distributed optimization algorithm that mitigates the impact of all three issues. HGO achieves this by using a scheme for workload distribution adaptive to devices, depending on their computational capabilities. The adaptation tackles both computational and data heterogeneity, thereby resulting in a more efficient use of available computing resources and improved learning accuracy, respectively. To tolerate Byzantine devices, HGO makes use of a robust *aggregation rule* (e.g., Trimmed Mean [27], MeaMed [26], Krum [4], or Aksel [6]).

Summary of Key Contributions: HGO is inspired by the stochastic *block coordinate descent* method. In particular, the server initiates each iteration by randomly choosing a subset of workers, and sending them its current estimates of the learning parameters. Then, an honest (non-Byzantine) worker sends a fragment (i.e., some coordinates) of its mini-batch stochastic gradient. The server identifies a block of coordinates for which it receives values from a quorum of workers, and applies the robust aggregation rule on this block of workers' gradients to update its current parameter estimates. We show linear and sub-linear convergence of HGO in the strongly convex and non-convex settings, respectively. Moreover, we also report on the learning performance of HGO through simulations (in the full version of the paper, we also present a deployment on actual Android-based smartphones). Our key theoretical and empirical findings are summarized below.

1. The worst case time complexity of HGO is still better than the one of classical SGD.
2. We demonstrate for the first time a critical trade-off between the convergence quality (i.e., the rate and the Byzantine robustness) of a gradient-based distributed ML algorithm and the computation costs for the devices.
3. Our experiments highlight the fact that the convergence quality of HGO is positively correlated with the computation power of the network, thereby enabling any device to participate in training ML models according to its computation capabilities.

Our theoretical analysis also apply to a much larger family of gradient-based protocols (including CD, SCD, Block CD, GD, SGD) as well as that of robust aggregation rules (including Median, Trimmed Mean, Krum, Aksel, MeaMed). To our knowledge, none of these learning algorithms have been analyzed under the combination of issues we consider.

The rest of the paper is organized as follows. We present the model and preliminary concepts in Section 2. Section 3 describes the algorithm and its time complexity. Section 4 presents our theoretical analysis of HGO and a discussion on the trade-offs. Section 5 summarizes its empirical performance. Section 6 discusses additional related works, and Section 7 concludes the paper.

For space limitations, **we defer all the proofs, as well as the detailed description of our evaluation, to the appendix**

2 Preliminaries

We present in this section the model setting and introduce the concepts of heterogeneity and robust aggregation. We also state all assumptions used in our theoretical results.

2.1 Model setting

We consider a set on N data points $X = \{x_1, \ldots, x_N\}$ that is arbitrarily partitioned over n workers (e.g., smartphones) in a parameter-server architecture. 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 be trustworthy. However, some of the workers may be faulty, and their identity is a priori unknown [17]. Specifically, there are two types of workers:

- **Correct workers:** These are honest processes that correctly follow the instructions prescribed by the server. We represent the correct workers by set $\mathcal{C} \subseteq [n]$ where $[n]$ denotes the set $\{1, \ldots, n\}$.
- **Byzantine workers:** The remaining workers, i.e., $[n] \setminus \mathcal{C}$, are assumed to be Byzantine, i.e., they have a totally arbitrary behavior. They may either crash, or inject adversarial perturbations in the system (e.g., see [4]).

[2] Please refer to Section 6 for additional related work.

[3] Abbreviation for *Heterogeneous Gradient-based Optimization*. HgO is also the chemical formula for mercuric oxide, a compound used as an antiseptic in medicine, and as a fungicide in agriculture.

Our goal is to design a distributed learning algorithm that allows all the correct workers to learn an optimal learning model Π over their set of 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 Π parameterized by $w \in \mathbb{R}^d$ for which each data point x incurs a loss of $l(w,\, x)$, where the function $l : \mathbb{R}^d \times X \to \mathbb{R}$ is assumed differentiable. The algorithm aims to find a local minimum of the function

$$\mathcal{L}(w) := \frac{1}{|X_{\mathcal{C}}|} \sum_{x \in X_{\mathcal{C}}} l(w,\, x). \tag{1}$$

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 classical algorithm of distributed stochastic gradient descent (SGD).

Distributed SGD Algorithm: This is an iterative algorithm where the server maintains an estimate of a solution to the learning problem, which is updated iteratively with the help of the workers. The initial estimate w_1 may be chosen arbitrarily. In each iteration $k = 1,\, 2, \ldots$, the server broadcasts its current estimate w_k to all workers, and waits a certain amount of time for the workers to return their stochastic gradients. Each worker i samples a random *mini-batch* $\zeta_{i_k} \subset \mathcal{S}_i$ of size s_i to compute the corresponding *stochastic* gradient:

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

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

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

where $\gamma \in \mathbb{R}_{>0}$ is the learning rate. Under certain conditions, the algorithm eventually outputs a solution to the learning problem. However, the convergence of the above algorithm often relies on the assumption that all workers have comparable computational capabilities.

2.2 Computational and data Heterogeneity

In addition to faults, when deploying distributed learning on low-powered hand-held devices, such as tablets or smartphones, workers may have very different computational capabilities. Specifically, we consider that each honest (or correct) worker i computes a single coordinate of gradient $\nabla l(w,\, x)$, for a given parameter w and data point $x \in \mathcal{S}_i$, with a rate of $\lambda_i \in \mathbb{R}_{>0}$. Therefore, if the server allocates τ amount of time to each worker in an iteration k, worker i computes $\min\{\frac{\lambda_i \tau}{s_i},\, d\}$ coordinates of its stochastic gradient $G_i(w_k)$. That is, 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 $\mathcal{Q}_j$ the set of workers that computed coordinate j of their stochastic gradients.

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 $\nabla\, \mathcal{L}(w)$, illustrated in Figure 1 below.

2.3 Robust Aggregation

In presence of Byzantine workers, the server cannot rely on the simple averaging of all the workers' gradients as in the classical distributed SGD algorithm. Rather it uses a coordinate-wise robust *aggregation rule* AR, which we formally define as follows in our context. For a vector $v \in \mathbb{R}^d$, its j-th element is denoted by $\{v\}_j$. Recall that $\mathcal{C}$ denotes the set of correct workers , and the function $\mathcal{L}(w)$ is defined in (1) above. For a correct worker $i \in \mathcal{C}$, recall that its gradient $G_i(w)$ at any parameter $w \in \mathbb{R}^d$ is defined by (2). The gradients of a Byzantine worker i may be arbitrary.

Definition 1. *For a given $q \in [d]$, $w \in \mathbb{R}^d$, an aggregation rule* AR $: \mathbb{R}^q \to \mathbb{R}$ *is* $\Delta(q, f)$-robust *if for every subset $\mathcal{Q} \subseteq [n]$ with $|Q| \geq q$,*

$$\mathbb{E}\left[\left(\text{AR}\left(\{G_i(w)\}_j;\; i \in \mathcal{Q} \right) - \{\nabla\, \mathcal{L}(w)\}_j \right)^2 \right] \leq \Delta(q, f) \left(\frac{1}{|\mathcal{C}_{\mathcal{Q}}|} \sum_{i \in \mathcal{C}_{\mathcal{Q}}} \sigma_i^2(w) \right),\; \forall j \in [d]$$

where $\mathcal{C}_{\mathcal{Q}} = \mathcal{C} \cap \mathcal{Q}$, and $\sigma_i^2(w) = \mathbb{E}\left[\left(\{G_i(w)\}_j - \{\nabla\, \mathcal{L}(w)\}_j \right)^2 \right]$ for all $i \in \mathcal{C}$. The robustness coefficient $\Delta(q, f)$ is a bounded positive real value that depends upon f and q.

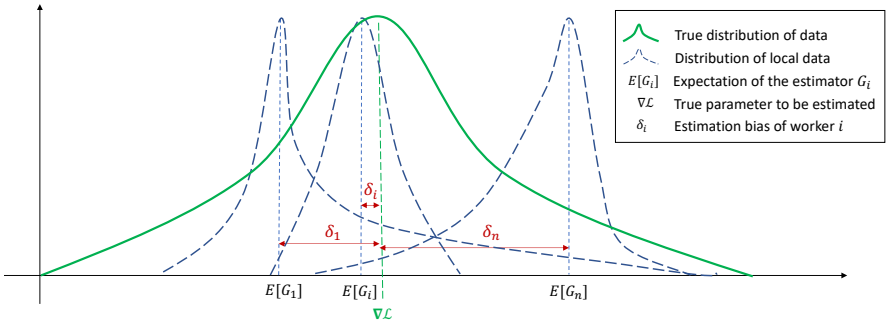

Figure 1: The dashed blue curves model the distributions of local data as seen by each worker. The green curve models the true distribution of data. In our setting, each worker tries to estimate a block of partial derivatives of the true cost function using only its local data. Due to data heterogeneity,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

Remark 1. *It is reasonable to suppose that increasing the number of correct workers reduces the error of a robust aggregation rule. Therefore, the robustness coefficient $\Delta(q, f)$ should be inversely related to q, for a fixed f. While earlier aggregation rules [4, 11] behaved against this intuition, new improved aggregation rules [27, 30, 6, 26] have robust coefficients that are either constant or decrease when the q increases. We only consider aggregation rules that satisfy this property.*

2.4 Assumptions

To formally analyze the convergence of our algorithm, we make the following standard assumption on the smoothness of the loss function [5]. For a vector v, we let $\{v\}_i$ denote its i-th element.

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 do present a stronger convergence guarantee 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 formally model the inherent inaccuracies of estimation,, we make the following two assumptions, that are satisfied in many learning problems [5, 23]. Note that we do not rely on the stronger assumption of uniformly bounded variance, which is indeed quite difficult to satisfy, and perhaps even impossible in the presence of strong convexity [23]. Instead, we 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 $w \in \mathbb{R}^d$,* $|\mathbb{E}_{x \sim \mathcal{S}_i} [\{\nabla l(w, x)\}_j] - \{\nabla \mathcal{L}(w)\}_j| \leq \delta_{i,j}$ *where $x \sim \mathcal{S}_i$ denotes uniform distribution 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 $j \in [d]$,* $\mathbb{E}_{x \sim \mathcal{S}_i} \left[(\{\nabla l(w, x)\}_j - \mathbb{E}_{x \sim \mathcal{S}_i} [\{\nabla l(w, x)\}_j])^2 \right] \leq M_w + Q_w \|\{\nabla \mathcal{L}(w)\}_j\|^2$.

3 Our Algorithm, and its Convergence Guarantee

We now present our proposed algorithm HGO.

The HGO algorithm, summarized in Algorithm 1, follows T steps of a *generalized* distributed SGD algorithm. At each iteration k, the server maintains an estimate w_k of an optimal parameter vector. To update its estimate, the server

Algorithm 1 HGO: **H**eterogeneous **g**radient-based **O**ptimization

Parameter Server

1: **Input:** q, AR, w_1, v, T
2: **Execute the following instructions in each step** $k = 1, \ldots, T$
3: $C_k \longleftarrow$ random permutation of set $\{1, \ldots, d\}$
4: Select q random workers $\{i_1, \ldots, i_q\} \subseteq \{1, \ldots, n\}$
5: Broadcast (w_k, C_k) to the selected q workers
6: Wait τ time units to receive workers' gradients
7: $V_{i_1}, \ldots, V_{i_q} \longleftarrow$ partial derivatives sent by the q workers, i.e., elements of their gradients at w_k
8: $R \longleftarrow$ sort $\big(|V_{i_1}|, \ldots, |V_{i_q}|,$ decreasing order$\big)$
9: $b_v \longleftarrow v^{\text{th}}$ item in the set R
10: $\mathcal{B}_k \longleftarrow$ first b_v elements in C_k
11: $\forall j \in \mathcal{B}_k$, identify the set $\mathcal{Q}_j \subseteq \{i_1, \ldots, i_q\}$ of workers that sent the j-th coordinate of their gradients.
12: Compute the aggregate of workers' partial derivatives $AG(w_k) \in \mathbb{R}^d$ such that for all $j \in [d]$,

$$\{AG(w_k)\}_j = \begin{cases} \text{AR}\left(\{V_i\}_j,\ i \in \mathcal{Q}_j\right) & , \quad j \in \mathcal{B}_k \\ 0 & , \quad \text{otherwise} \end{cases}$$

where $\{V_i\}_j$ is sent by worker i for the j-th element of its gradient.
13: Update the parameter vector to $w_{k+1} = w_k - \gamma AG(w_k)$

Honest Worker i

1: **Input:** local dataset S_i, mini-batch size s_i, computation rate λ_i
2: **If** the broadcast message (w_k, C_k) is received **then** execute the following steps:
3: Sample a random mini-batch ζ_{i_k} of size s_i from dataset S_i
4: $\mathcal{B}_k^i \longleftarrow$ first b_i elements in C_k where $b_i = \min\{\frac{\tau\lambda_i}{s_i}, d\}$
5: Construct the set $V_i = \big(\{G_i(w_k\}_j, j \in \mathcal{B}_k^i\big)$ where $G_i(w_k)$ is as defined in (2)
6: Send back to the Server V_i

chooses a random order of coordinate indices C_k (i.e., C_k is a permutation of the set $\{1, \ldots, d\}$), selects a random set of q workers, and broadcasts (w_k, C_k) to these workers. Then, the server waits for τ time units for workers' responses. 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 $G_i(w_k)$ 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. However, a Byzantine worker may arbitrary values for its partial derivatives. After τ 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, described in steps 8, 9 and 10 of Algorithm 1. The server then applies a robust *aggregation rule* AR (defined above in Definition 1) on the consistent coordinates of the gradients sent by the workers, and uses the outputs to update its current parameter vector w_k.

Remark 2. *We do not force the workers to complete the whole training cycle, for this can last longer than the average battery can allow. Therefore, a worker can join the training, disconnect at timestamp k_1, then reconnect at timestamp k_2 (after a battery, network or hardware issue). The server keeps track of all the active devices in each iteration. The algorithm works as long as a quorum of workers are active at each iteration. The number of Byzantine workers that can be tolerated inside this quorum is dictated by the aggregation rule.*

Time complexity of HGO. We give now the total time complexity of the algorithm:

Proposition 1 (HGO time complexity)**.** *Let $\bar{\lambda}$ be the average computation rate and $\bar{s}$ the average mini-batch size of the workers. Let C_{pd} denote the cost of computing one partial derivative of a stochastic gradient. If* Trimmed Mean *is used as an aggregation rule, then the total time complexity of* HGO *running for T iterations is $\mathcal{O}\left(T\frac{\bar{\lambda}\tau}{\bar{s}}(C_{pd} + q\log(q))\right)$ on average, and $\Omega\left(Td(q^2 + C_{pd})\right)$ at worse.*

Proof. Let us consider the computations at the server level. At the worst case, the q random workers will all be powerful workers and thus able to compute the full gradients. Therefore, upon receiving the gradients, computing the cardinal of the received sets V_i's is $\Omega(qd)$. On average, this same step is done in $\mathcal{O}(qb)$, where $b = \frac{\bar{\lambda}\tau}{\bar{s}}$ is the average block size, given an average computation rate $\bar{\lambda}$ and an average mini-batch size $\bar{s}$. In both cases, the sorting is done on q values. Using *QuickSort*, the worst case complexity is $\Omega(q^2)$ and the average complexity is $\Omega(q\log(q))$. Finally, considering

that applying *Trimmed Mean* on q values has an average complexity of $\Omega(q\log(g))$ and a worst complexity of $\Omega(q^2)$, the aggregation step of the algorithm will be in $\mathcal{O}(q\log(q)\frac{\bar{\lambda}\tau}{\bar{s}})$ on average and at worse, $\Omega(q^2 d)$. On the other hand, each worker i computes a number of partial derivatives of its stochastic gradient, depending on its computation rate λ_i, the amount of time available τ and the mini-batch size s_i. Let C_{pd} be the cost of computing one partial derivative of the stochastic gradient. Then, the average time complexity at the worker level is $\mathcal{O}(\frac{\bar{\lambda}\tau}{\bar{s}}C_{pd})$, and at worse $\Omega(dC_{pd}$. Thus, full time complexity of the algorithm at each iteration is $\mathcal{O}\left(\frac{\bar{\lambda}\tau}{\bar{s}}(C_{pd}+q\log(q))\right)$ and at worse $\Omega\left(d(q^2+C_{pd})\right)$, which concludes the proof. □

4 Theoretical Guarantees

Inspired by prior works in the homogeneous setting [5, 4, 6, 30], we show that HGO eventually (when $T\to\infty$) outputs an estimate w_k such that

$$\|\nabla\mathcal{L}(w_k)\|^2 \leq \frac{d\Delta(v,f)A_{w_k}}{b\left(1-\Delta(v,f)B_{w_k}\right)}$$

where $b=\lim_{T\to\infty}\min_{k=1,\ldots,T}|\mathcal{B}_k|$,

$$A_w := \frac{1}{|C|}\sum_{j=1}^{d}\sum_{i\in\mathcal{C}}\left(\frac{M_w}{s_i'}+\delta_{i,j}^2\right),\ B_w := \frac{1}{|C|}\sqrt{\sum_{j=1}^{d}\left(\sum_{i\in\mathcal{C}}\frac{Q_w}{s_i'}\right)^2}\ \text{ with }\ s_i' := \frac{s_i(|S_i|-1)}{|\mathcal{S}_i|-s_i}. \tag{4}$$

Informally, we show that our algorithm eventually reaches a ball centered at a local optimum, whose radius is a function of the statistical error. Figure 2 below illustrates the trajectory of the algorithm in the context where there exists a unique optimal solution w^*.

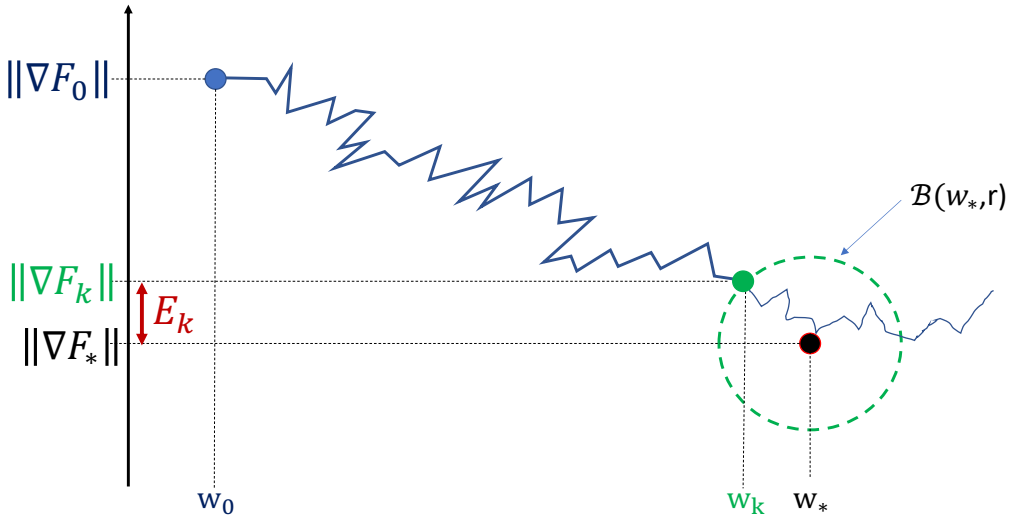

Figure 2: As long as the the parameter vector w_k is inside $\mathcal{W}$, which means that the gradient norm is still big enough, the algorithm will follow a descent direction, towards a ball $\mathcal{B}(w_*,r)$ centred at a local optimum, with a radius r, which is a function of the statistical error. When the condition is no longer satisfied, the noise (induced by correct and Byzantine workers) will prevent progress towards the optimum.

We present below the formal convergence result of our algorithm. First, we divide the parameter space into two regions $\mathcal{W}$ and $\mathbb{R}^d \setminus \mathcal{W}$ such that

$$\mathcal{W} := \left\{ w \in \mathbb{R}^d;\ \min_{\mathcal{B} \subseteq [d]} \left\{ (1 - \Delta(v,f)B_w) \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 - \Delta(v,f)A_w \right\} > 0 \right\}. \tag{5}$$

Henceforth, we will write $\Delta(v,f)$ simply as Δ.

Remark 3. *The condition* $(1 - \Delta B_w) \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 - \Delta A_w > 0$ *is a generalization of the* variance-norm ratio *used in the very first work addressing SGD in a Byzantine environment for ensuring* (α, f)*-Byzantine resilience of their proposed aggregation rule (*KRUM*), and thereby convergence of their algorithm in the presence of Byzantine devices [4]. They only considered the special case of SGD (i.e.,* $b = d$*) with unbiased estimation (i.e.,* $\forall (i,j) \in [N] \times [d], \delta_{i,j} = 0$*), uniform upper bound on the variance (i.e.,* $Q_w = 0$*), and a mini-batch of size unity (i.e.,* $\forall i \in [n], s_i = 1$*). In this special case, our generalized condition reduces exactly the same condition as in Proposition 1 of their work. The authors also claim that their condition can be satisfied when using larger mini-batch sizes. Our formulation proves 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.*

4.1 Convergence analysis

We note that it is impossible to guarantee progress towards a local optimum in any step k if the direction of the aggregated gradient $AG(w_k)$ is opposite to the actual descent direction of the loss function necessary. Specifically, the scalar product between the aggregated gradient $AG(w_k)$ and the gradient $\nabla \mathcal{L}(w_k)$ must be positive. We show below in Lemma 1 that this condition holds true in our setting when $w_k \in \mathcal{W}$ and the aggregation rule AR is Δ-robust.

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

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

Proof. (Sketch) We ignore the subscript k and write $\Delta(v,f)$ simply as Δ. First, using Definition 1 and Assumptions 3 and 4, we derive an upper bound on the error of the aggregated vector $\mathbb{E} \left\| AG(w) - [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 \leq r^2$ where $r^2 = \Delta(v,f) \left(A_w + B_w \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 \right)$. This implies that $\left\| \mathbb{E}[AG(w)] - [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 \leq r^2$ (by Jensen's inequality). Thus, $\mathbb{E}[AG(w)]$ belongs to a ball centered at $[\nabla \mathcal{L}(w)]_{\mathcal{B}}$ with radius r. Then, we can show, using trigonometric reasoning that $\mathbb{E}[\langle AG(w), [\nabla \mathcal{L}(w_k]_{\mathcal{B}_k} \rangle] > (1 - \sin(\theta)) \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2$ where $\sin(\theta) = \frac{r}{\left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|}$. The rest follows from algebraic calculations. □

Using the above result we obtain the convergence properties of HGO for strongly convex and non-convex functions. In the following theorem we show linear convergence of HGO under strong convexity. Under Assumption 1, for a block of coordinates $\mathcal{B}$ we define $L_{\mathcal{B}} = \sum_{j \in \mathcal{B}} L_j$.

Theorem 1 (Strongly convex)**.** *Suppose that Assumptions 1, 2, 3 and 4 hold true. Consider Algorithm 1 with a constant learning rate* $\gamma < \min_{k \in [T]} \left\{ \frac{1}{L_{\mathcal{B}_k}} \right\}$*. If* $\Delta B_{w_k} < 1$ *for all* $k \in [T]$ *then*

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

where $\mathcal{L}^*$ *denotes the minimum value of* $\mathcal{L}(w)$*,* $b = \min_{k \in [T]} |\mathcal{B}_k|$*,* $A = \max_{k \in [T]} A_{w_k}$*,* $B = \max_{k \in [T]} B_{w_k}$*, and* $H = \frac{d \Delta A}{2b\mu(1 - \Delta B)}$*.*

Proof. (Sketch) Under Lipschitz smoothness, i.e., Assumption 1, we have

$$\mathcal{L}(w_{k+1}) \leq \mathcal{L}(w_k) - \gamma_k \langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, AG(w_k) \rangle + \gamma_k^2 \frac{L_{\mathcal{B}_k}}{2} \left\| AG(w_k) \right\|^2$$

Then, the term $\left\| AG(w_k) \right\|^2$ can be decomposed using $\left\| [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} - AG(w_k) \right\|^2 = \left\| [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \right\|^2 - 2\langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, AG(w_k) \rangle + \left\| AG(w_k) \right\|^2$ in order to exploit the result from Lemma 1. After introducing the total expectation and rearranging terms, we can get to an inequality of the form:

$$\mathbb{E}\, \mathcal{L}(w_{k+1}) \leq -X \left\| \nabla \mathcal{L}(w_k) \right\|^2 + Y$$

Using Assumption 2 and rearranging the terms, we obtain a contraction inequality:

$$\mathbb{E}\,\mathcal{L}(w_{k+1}) - \mathcal{L}(w_*) - H \leq (1-\kappa)\left(\mathbb{E}\,\mathcal{L}(w_k) - \mathcal{L}(w_*) - H\right)$$

If repeated through iterations $(1, \ldots, T)$, we obtain the desired result. □

As known in the literature, it is hard to come up with a strong convergence result for gradient-based algorithms (especially for CD [25, 21]) in non-convex settings. We show instead that the expected squared norm of at least one of the gradients observed in the sequence $\nabla\,\mathcal{L}(w_1), \cdots, \nabla\,\mathcal{L}(w_T)$ after T iterations will upper bounded.

Theorem 2 (Non-convex)**.** *Suppose that Assumptions 1, 3 and 4 hold. If $\Delta B_k < 1$ and $\gamma_k < \frac{1}{L_{\mathcal{B}_k}}$, then we have:*

$$\min_{1<i<T} \mathbb{E}\left[\left\|\nabla\,\mathcal{L}(w_i)\right\|^2\right] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{Tb\gamma(1-\Delta B)} + \frac{d\Delta A}{b(1-\Delta B)}$$

where $A = \max_{k\in[T]} A_{w_k}$, $B = \max_{k\in[T]} B_{w_k}$, $\gamma = \min_{k\in[T]} \gamma_k$ and $b = \min_{k\in[T]} b_k$.

Proof. (Sketch) Using the same arguments from the previous proof, we can obtain the following inequality:

$$\mathbb{E}\,\mathcal{L}(w_{k+1}) \leq -X\left\|\nabla\,\mathcal{L}(w_k)\right\|^2 + Y$$

Introducing the total expectation, rearranging the terms and averaging the gradients through $(1, \ldots, T)$ gives:

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}\left\|\nabla\,\mathcal{L}(w_i)\right\|^2\right] \leq \frac{\kappa}{T}(\mathcal{L}(w_1) - \mathcal{L}(w_*)) + H \tag{6}$$

Since $\min_{1<i<T} \mathbb{E}\left[\left\|\nabla\,\mathcal{L}(w_i)\right\|^2\right] \leq \mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}\left\|\nabla\,\mathcal{L}(w_i)\right\|^2\right]$, we obtain the desired result. □

About the learning rate. In the two theorems, we can see that increasing the learning rate γ improves the convergence rates. On the other hand, the learning rate γ_k at each iteration must satisfy the condition $\gamma_k < \frac{1}{L_{\mathcal{B}_k}}$, where $L_{\mathcal{B}_k}$ is the Lipschitz constant of the truncated gradient $[\nabla\,\mathcal{L}(w_k)]_{\mathcal{B}_k}$. Since the block $\mathcal{B}_k$ constructed by the server varies through iterations, this upper bound on the learning rate also fluctuates. In practice, choosing a small constant learning rate $\bar{\gamma}$ such that $\bar{\gamma} < \min_{k\in[T]} \frac{1}{L_{\mathcal{B}_k}}$ is a safe choice. However, to improve the convergence rate, one could try to reach this upper bound at each iteration by using a dynamic learning rate γ_k, inversely proportional to $\sqrt{|\mathcal{B}_k|}$.

Remark 4. *Our work can be seen as a generalization of previous works on Byzantine-resilient optimization, encapsulating a large family of descent algorithms (CD, SCD, Block CD, GD, SGD). As a matter of fact, many works can be summarized within our framework. It suffices to replace the variables in our result (depending on the set of assumptions, the algorithm and the defense mechanism) to get the convergence properties. For instance, in [25], 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 descent direction. In this simple algorithm, the descent direction 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 (no estimation), there will be no variance ($M_k = Q_k = 0$), no aggregation ($\Delta(q, f) = 0$), 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 $A_w = B_w = 0$ and $H_k = 0$. The author also uses a loose bound on the learning rate $\gamma < \frac{1}{dL_{max}}$. Replacing these values in Theorem 1 recreates the exact convergence theorem of RCD, presented in the second part of Theorem 1 in [25].*

4.2 Discussing the trade-offs

Our theoretical results formalize some trade-off relationships between the convergence properties, the behavior and the computation power of the workers, as well as the nature of the dataset's distribution. We discuss some of them in the following.

4.2.1 Convergence properties vs time complexity

The robustness coefficient $\Delta(q, f)$ introduced in Definition 1 and used in our results is a key factor of the aggregation error. The value of the robustness coefficient indeed depends on the quality of the aggregation rule we use, but also on the number of Byzantine workers within the selection of random workers. In fact, for a fixed estimation of the number f of Byzantine workers, increasing the number q of random workers selected at the beginning of each iteration increases the number of correct workers in this random set. This also means that the Byzantine impact is reduced by making $\Delta(q, f)$ smaller. 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. This simply means that it takes less time for this term to decay, therefore implying a faster convergence rate. The statistical error is also positively correlated to this coefficient. However, as presented in Proposition 1, the time complexity of the algorithm also increases with q. In short, better convergence properties imply a bigger time complexity.

4.2.2 Heterogeneity vs convergence properties

From the theorems, we can see 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 b_k of the block $\mathcal{B}_k$ constructed by the server at iteration k. In fact, increasing $b = \min_k b_k$ also increases the convergence rate and reduces the statistical error. There are three ways to increase b. We can either decrease the minimum number of workers v executing the aggregation, but this quantity is lower-bounded by $\frac{f}{\alpha}$, where α is the ratio of Byzantine workers. We can also decrease the mini-batch sizes, so that every worker dedicates its resources to computing more blocks ($\lambda_i \tau = b_i s_i$); but we would rather increase the mini-batch sizes, because of their positive effect on the convergence properties. Finally, the server can only contact the powerful workers (the ones with large computation rates λ_i) in order to increase the block size and the mini-batch size, improving the convergence rate and the statistical error. However, doing this may exclude workers with valuable data from the learning procedure, which certainly leads to a worse statistical error. To sum up, improving the workers' hardware specifications also improves the convergence properties.

4.2.3 Byzantine tolerance vs hardware heterogeneity

The condition $\Delta B_k < 1$ was stated in the two theorems in order to guarantee convergence. This inequality formalizes the trade-off between the Byzantine tolerance (through the robustness coefficient) and the computation power profile of the workers (captured in the variable B_k). To tolerate a bigger Byzantine impact, B_k must decrease in order to follow a descent direction at step k. Decreasing B_k implies either to decrease the size b_k of the block $\mathcal{B}_k$, or to increase the size of the mini-batches selected by the workers. 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. When the computation rates are high, it will be possible to increase both b_i and s_i, in order to both improve the convergence properties and satisfy the condition. In other words, a network full of weak devices cannot handle a strong Byzantine impact.

5 Evaluation

In this section, we evaluate the performance of HGO against the classical SGD. We also conduct experiments to demonstrate the trade-offs we discussed in Section 4.2. Due to page limitation, we only present a few experiments that support our theoretical analysis. The source code, as well as a complete set of experiments, are available in the following GitHub repository: `https://anonymous.4open.science/r/HgO-5136`

5.1 Environment setting

We train a binary logistic regression model and a feed forward neural network using the ***MNIST*** dataset. We consider workers with different computation rates and use the categories "weak", "average" and "powerful" to distinguish the hardware specifications of workers. Each category represents a set of workers whose computation rates are normally distributed over a predetermined mean value. In all the experiments, we assume that the total number of workers n contains 30%, 40% and 30% of weak, average and powerful devices, respectively. In one particular experiment, we construct scenarios where these percentages vary, starting from C_{weak} (where all the workers are weak devices) to $C_{powerful}$ (where all the workers are powerful). We evaluate the performance of HGO when coupled with different aggregation rules, against a varying number of Byzantine workers implementing two state-of-the-art attacks: ***Little Is Enough*** [3] and ***Fall Of Empires*** [28]. These attacks basically exploits the normal distribution of correct gradients to construct malicious estimates that are still within the correct range, thus bypassing classical filters based on evaluating distances. We consider a homogeneous partitioning of the dataset over the workers, unless specified otherwise.

5.2 Comparing HGO and SGD

Unlike our algorithm, the classical SGD aggregates the gradients estimated by all the workers. Therefore, the server may be waiting for the weakest device to complete its computations. The waiting time might be infinite when a single device cannot handle the workload (cpu or memory limits). When setting a a fixed waiting time τ, SGD basically discards the weakest workers, which results in partial learning, impacting negatively the final accuracy. This is further accentuated when datasets are not identically distributed. Figures 3 and 4 shows that HGO has improved convergence properties compared to SGD.

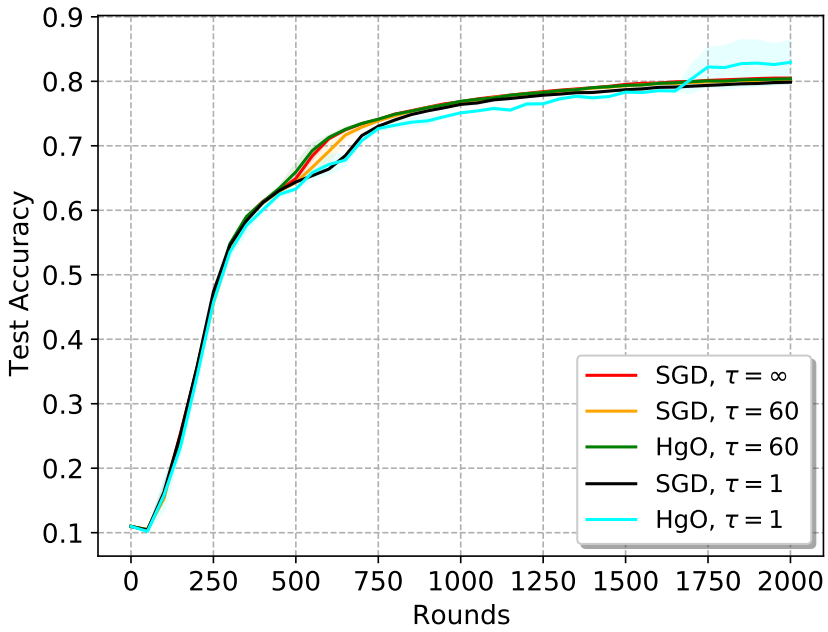

Figure 3: Training a feed forward neural network on MNIST using $n = 50$ workers in an honest setting. The server averages the estimates from q random workers at each iteration. While all algorithms reach nearly the same accuracy, it should be noted that, in terms of run time, HGO is clearly faster than SGD($\tau = \infty$) but slower than SGD($\tau < \infty$), due to the overhead induced when constructing the blocks to update. This should not be the case when using more complex models and datasets.

5.3 Byzantine impact vs convergence properties

In our results, the Byzantine impact is captured in the variable $\Delta(v, f)$. This robustness coefficient has a direct impact on the convergence rate as well as on the statistical error of the algorithm. Its value depends on the number of Byzantine workers, on their strategies, and also on the weakness of the aggregation rule we use. We illustrate this concept in Figures 5, 6 and 7.

5.4 Varying the hardware profile of the network

In the next set of experiments, we show how the hardware profile of the network has an impact on the convergence properties. Particularly, we show that introducing only 10% of powerful workers in an all-weak network (scenario C_{min}) substantially improves the convergence properties (see Figure 8).

6 Related Work

Many works have studied the Byzantine resilience of gradient-based descent optimization algorithms [4, 6, 8, 30, 7, 2, 9, 29] by leveraging robust statistics, filtering schemes or coding theory to defend against a fraction of Byzantine workers.

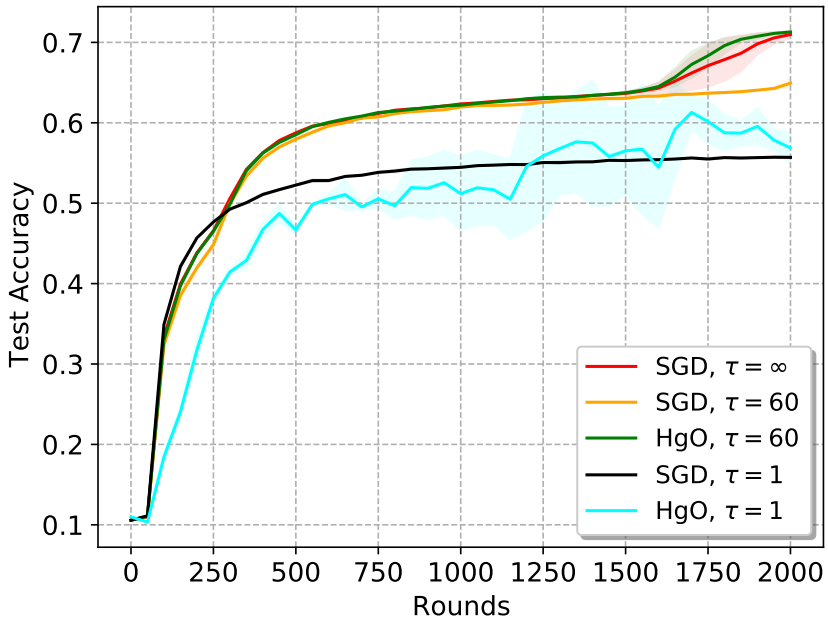

Figure 4: Training a feed forward neural network on a non-identically distributed version of MNIST, using $n = 50$ workers in an honest setting. The server averages the estimates from q random workers at each iteration. Note that HGO($\tau = 60$) is performing as well as the much slower SGD($\tau = \infty$), runtime-wise. Also, because the dataset is not identically distributed, SGD($\tau < \infty$), discards weak workers that may have valuable data, as opposed to HGO which is aggregating information from all the workers, regardless of their computation capabilities. Therefore, HGO has a better accuracy than SGD($\tau = \infty$).

However, they all assume unbiased estimation and i.i.d. local datasets. Particularly, [30] analyzed the Byzantine resilience of gradient descent coupled with trimmed mean and median, but the analysis only applies to a small number of Byzantine workers (very far from optimal) and, most importantly, it relies on strong assumptions on the distribution of the gradients (bounded skewness and sub-exponential distribution). A work closely related to ours is [9]. In the proposed synchronous coordinate descent algorithm, an encoding scheme is used to defend against Byzantine workers, which is not suitable for federated learning architectures due to privacy concerns. To minimize overhead costs, the condition on the maximal number of Byzantine workers falls to $n > 3f$, which is not optimal. Besides, the encoding time is high, and depends on the dimension of the problem ($\mathcal{O}\left(d\left(\frac{\epsilon}{1+\epsilon}n + 1\right)\right)$) as well as the quadratic cost of iteration at the PS ($\mathcal{O}(n^2)$).

On the other hand, some papers [14, 15, 18] studied SGD with local iterations on heterogeneous data, without assuming Byzantine workers. [10, 24] analyzed distributed SGD assuming data heterogeneity and Byzantine attacks. However, they use encoding schemes, which are not practical in our setup. Finally, [12] used clustering combined with robust statistics to transform the heterogeneous dataset into homogeneous clusters before the distributed optimization step. Besides the fact that this algorithm uses an extra step (clustering) to handle heterogeneity, which impacts its time complexity negatively, reassigning data to clusters rises some privacy concerns. A common modeling of heterogeneity in all these works is to assume a dissimilarity at the optimum between the local loss functions, gradients or parameter vectors.

Most importantly, none of these works consider the heterogeneity of the workers in terms of computation capabilities. The closest work to ours is [13]. The authors propose to handle the effect of stragglers in a heterogeneous network of workers, but using a smart learning rate schedule. However, the workers are still using the same optimization algorithm (SGD). Besides, Byzantine resilience and data heterogeneity are not analyzed.

In our paper, we study for the first time the simultaneous execution of a large family of gradient-based algorithms (CD, BCD, SCD, GD, SGD) under the supervision of a parameter server, using a generic defense mechanism. Our

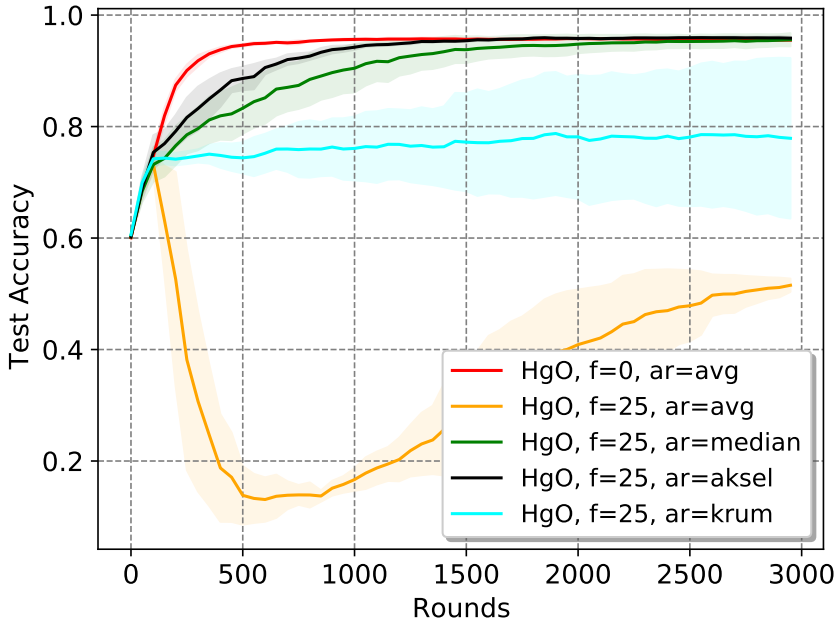

Figure 5: Training a binary logistic regression on MNIST, with a total of $n = 100$ workers. Among the $q = 80$ random workers contacted by the server at each iteration, $f = 25$ are Byzantine and implementing the attack [28]. As a benchmark, we also plot the accuracy of HGO using *Average* with no attack. While *Average* is not resilient to Byzantine attacks, *Krum*, *Median* and *Aksel* are able to defend against this attack with different convergence properties. Particularly, *Aksel* has the highest convergence rate and statistical error.

algorithm provably converges under a large set of constraints: Byzantine faults, stale and non-fully active workers, data heterogeneity and hardware heterogeneity. We also use a different approach to model data heterogeneity. As a matter of fact, we model all kinds of inaccuracies at the worker level (of which heterogeneity is an example) using a coordinate-wise bias on partial derivatives, at each iteration, which is different from assuming dissimilarity at the optimum. Moreover, to our knowledge, we are the first to study the trade-offs between the convergence rate, the statistical error, the Byzantine resilience, the per-iteration cost and the time complexity of the algorithm.

7 Conclusion

We present HGO, a new generic algorithm that can be equipped with any defense mechanism, and whose iteration cost can be tuned to match the capabilities of workers in a parameter server architecture for smartphone machine learning. We prove convergence for strongly convex as well as non-convex cost functions under a non-trivial combination of hypotheses, namely: Byzantine failures, semi-asynchrony, biased estimation and non-uniformly bounded variance. We study, for the first time, gradient-based optimization in the presence of Byzantine workers without requiring the remaining honest workers to be fully accurate. We also demonstrate trade-offs between important quantities like the statistical error rate, the impact of Byzantine workers, the level of inaccuracy of honest workers, the iteration cost and the convergence speed. Empirically, we show that the lightest instance of HGO, updating only one coordinate and using only one data point at each iteration, is able to converge for simple ML models. We also demonstrate the superiority of hybrid versions of HGO over classical SGD, in the case of smartphone ML, in terms of iteration cost as well as convergence speed. Our algorithm is also robust against state-of-the-art attacks. Ethically, our work encourages data privacy by making smartphones (which are the most important personal data carrier nowadays) contribute directly to AI projects from big companies, without exposing local data to third parties.

HGO only exchanges a block of coordinates instead of the full gradient at each iteration, which makes it better than gradient alternatives in terms of communication complexity. However, it is still considered as a network bottleneck in the case of smartphones connected to the internet. A better option would be to locally train for multiple steps, instead of

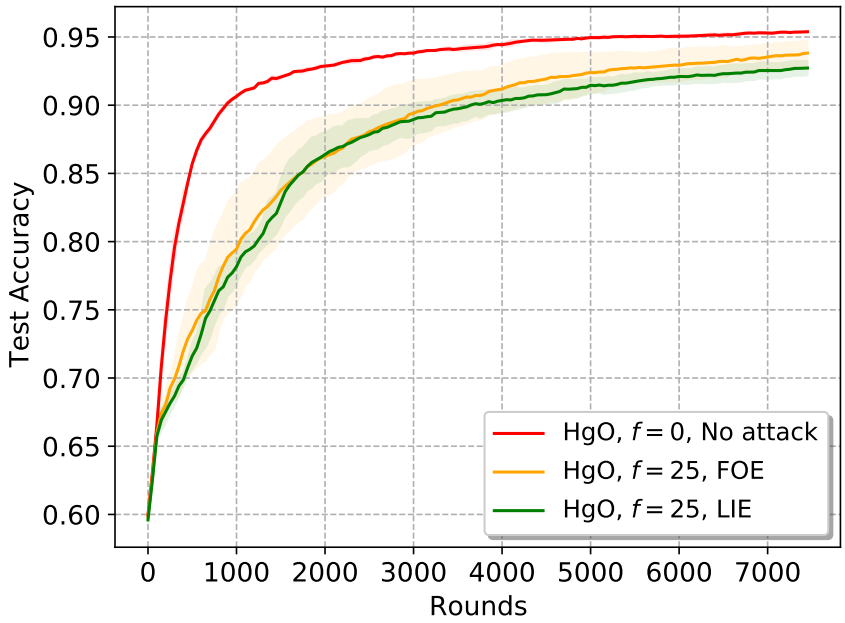

Figure 6: Training a binary logistic regression on MNIST, with a total of $n = 100$ workers. Among the $q = 80$ random workers contacted by the server at each iteration, $f = 25$ are Byzantine. As a benchmark, we also plot the accuracy of HGO using *Average* with no attack. In this experiment, the attack [3] is a little bit more devastating than [28]. This gap is further emphasized when using different ML models and different datasets.

one iteration at a time, before aggregating the parameter vectors. Also, while the assumptions on the variance and the bias of the estimation concern each data point and each coordinate (definitely stronger than the classical versions), it allowed us to integrate the mini-batch and the block analysis in our theoretical results. Perhaps a set of more relaxed assumptions would provide better or at least similar results.

Can this work be extended to non-smooth functions, or to second order optimization? What is the impact of the learning rate on the statistical error? Which computation power profile is needed to reach a target accuracy within a given time budget? Finally, what is the impact of network bandwidth on the convergence properties? We leave these questions open for future works.

References

[1] Martin Abadi et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. Byzantine stochastic gradient descent. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 4613–4623. Curran Associates, Inc., 2018.

[3] Gilad Baruch, Moran Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8635–8645. Curran Associates, Inc., 2019.

[4] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. NeurIPS'17, page 118–128, Red Hook, NY, USA, 2017. Curran Associates Inc.

[5] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

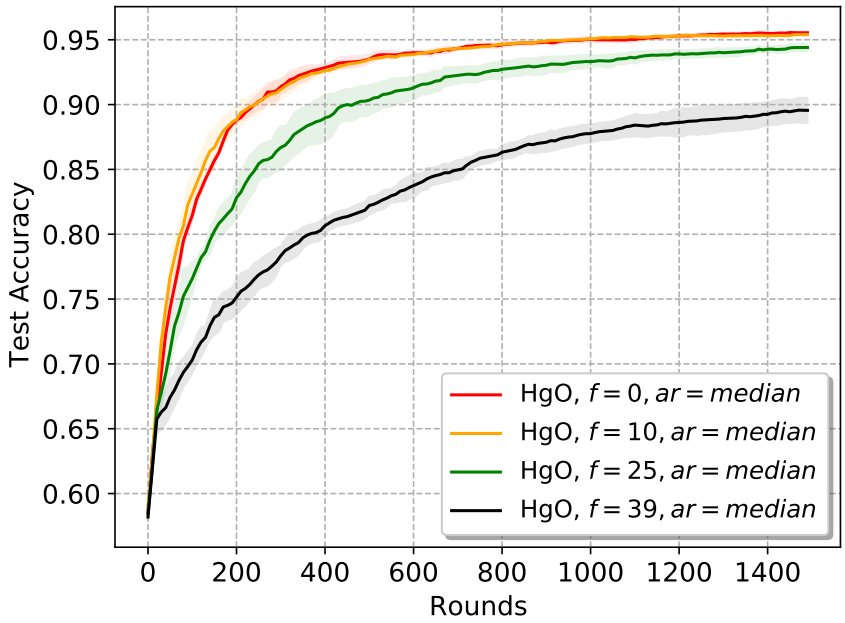

Figure 7: Training a binary logistic regression on MNIST, with a total of $n = 100$ workers. The server aggregates the partial gradients of $q = 80$ random workers at each iteration, using *Median*. We increase the number of Byzantine workers implementing the attack [28] ($f = 0, 10, 25, 39$) and compare the results with HGO using *Average*, under no attack. As expected, decreasing the number of Byzantine workers improves the convergence rate and the statistical error.

[6] Amine Boussetta, El-Mahdi El-Mhamdi, Rachid Guerraoui, Alexandre Maurer, and Sébastien Rouault. AKSEL: Fast Byzantine SGD. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, volume 184 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[7] Lingjiao Chen, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. Draco: Byzantine-resilient distributed training via redundant gradients, 2018.

[8] Georgios Damaskinos, El Mahdi El Mhamdi, Rachid Guerraoui, Rhicheek Patra, and Mahsa Taziki. Asynchronous Byzantine machine learning (the case of SGD). In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1145–1154, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[9] D. Data and S. Diggavi. Byzantine-tolerant distributed coordinate descent. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2724–2728, 2019.

[10] Deepesh Data, Linqi Song, and Suhas Diggavi. Data encoding methods for byzantine-resilient distributed optimization. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2719–2723, 2019.

[11] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in Byzantium. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3521–3530. PMLR, 10–15 Jul 2018.

[12] Avishek Ghosh, Justin Hong, Dong Yin, and Kannan Ramchandran. Robust federated learning in a heterogeneous environment, 2019.

[13] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. Heterogeneity-aware distributed parameter servers. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 463–478, New York, NY, USA, 2017. Association for Computing Machinery.

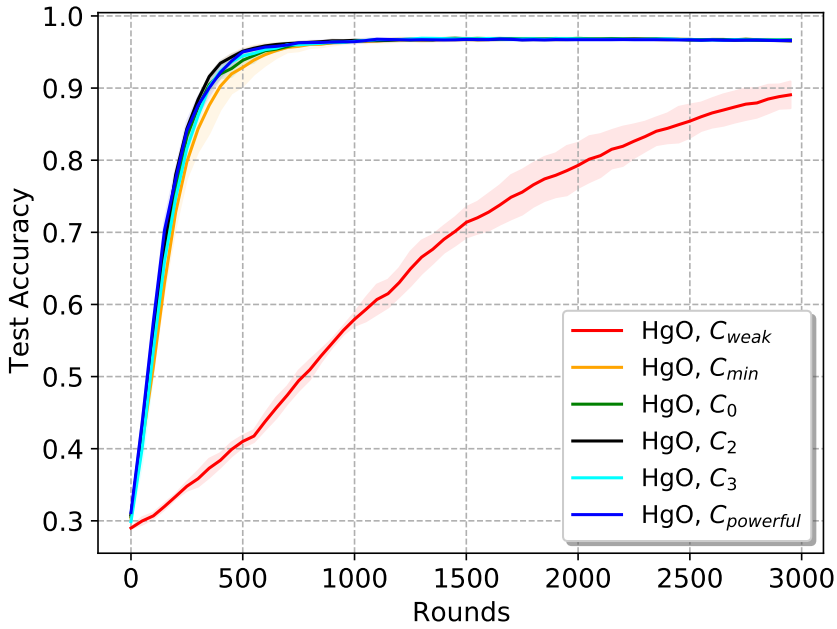

Figure 8: Training a binary logistic regression on MNIST, with a total of $n = 100$ workers in an honest setting. The server selects $q = 80$ random workers at each iteration, and averages their estimates. We plot the test accuracy of HGO under different hardware profiles. C_{weak} is a scenario with only weak devices. C_{min} replace 10% of the devices in C_{weak} by powerful workers. C_1, C_2 and C_3 are different combinations, increasing the proportion of average and powerful workers, respectively. $C_{powerful}$ is a network with only powerful devices. Interestingly, introducing a network hardware profile equivalent to C_{min} is sufficient to attain convergence properties similar to $C_{powerful}$.

[14] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtarik. Tighter theory for local sgd on identical and heterogeneous data. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 4519–4529. PMLR, 26–28 Aug 2020.

[15] Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian U. Stich. A unified theory of decentralized sgd with changing topology and local updates. In *ICML*, pages 5381–5393, 2020.

[16] Jakub Konecný, H. McMahan, D. Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *ArXiv*, abs/1610.02527, 2016.

[17] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[18] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2020.

[19] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for non convex optimization. *In NIPS, pages 2737–2745*, 2015.

[20] Alexander McFarlane Mood. Introduction to the theory of statistics. 1950.

[21] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[22] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition, 2014.

[23] Lam Nguyen, PHUONG HA NGUYEN, Marten van Dijk, Peter Richtarik, Katya Scheinberg, and Martin Takac. SGD and hogwild! Convergence without the bounded gradients assumption. volume 80 of *Proceedings of Machine Learning Research*, pages 3750–3758, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[24] Shashank Rajput, Hongyi Wang, Zachary B. Charles, and Dimitris Papailiopoulos. Detox: A redundancy-based framework for faster and more robust gradient aggregation. In *NeurIPS*, 2019.

[25] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[26] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Generalized byzantine-tolerant sgd, 2018.

[27] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Phocas: dimensional byzantine-resilient stochastic gradient descent, 2018.

[28] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation, 2019.

[29] Z. Yang and W. U. Bajwa. Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning. *IEEE Transactions on Signal and Information Processing over Networks*, 5(4):611–627, 2019.

[30] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates, 2018.

A Preliminaries

A.1 Notations

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
λ_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
τ	Number of time units the server waits for worker's estimations
AR	Robust aggregation rule
Δ	Robustness coefficient of AR
v	Minimum number of workers to execute the aggregation
μ	Strong convexity parameter
L_i	Coordinate-wise lipschitz parameter
w_k	Parameter vector at iteration k
d	Dimension of the parameter vector
$\mathcal{S}_i$	Local data set of worker i
ζ_i	Random mini-batch selected by worker i
s_i	$= \lvert\zeta_i\rvert$ mini-batch size
$\mathcal{B}_i$	Block of coordinates computed by worker i
b_i	$= \lvert\mathcal{B}_i\rvert$ 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
γ_k	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 $w \in \mathbb{R}^d$
F_0	$= \mathcal{L}(w_1) - \mathcal{L}(w_*)$ (initial gap)

A.2 Cost functions

Many fundamental problems in machine learning fall into the class of convex optimization. We consider the structured convex function $\mathcal{L}(w) = \sum_{i=1}^{m} g_i(w) + \lambda\Omega(w)$, where $g_1, \cdots, g_m$ and Ω are convex functions and $\lambda > 0$ is a fixed parameter. Using this function, one can model support vector machines, logistic regression, least squares, ridge and lasso regression, which are the most used for classification and regression to this day.

HGO is essentially exploiting the light iteration cost of coordinate descent methods combined with the one of stochastic gradient methods. To illustrate the cheaper cost of iteration (of CD with respect to GD), consider a classical least square problem: $\min_x \|Ax - b\|^2$, where $A \in \mathbb{R}^{m\times d}$ and $x \in \mathbb{R}^d$. If p denotes the average number of non-zeros in each column of the matrix A, then computing a partial derivative requires $\mathcal{O}(p)$ time units, while computing a gradient requires $\mathcal{O}(dp)$ time units. Although CD methods have been proved to converge and to be faster than corresponding gradient methods for this class of problems, we consider in our work a general cost function, and we prove different types of convergence for each class of functions.

B Proof of theoretical results

B.1 Proposition ??

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 $\text{TM} : \mathbb{R}^n \longrightarrow \mathbb{R}$ averages all but the t smallest and largest values among the n initial values where we assume that $t < n/2$. In [27], the authors derived, assuming *unbiased estimation*, an upper bound on the trimmed mean of n vectors 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 (2). We also denote the gradients sent by a Byzantine worker i by $G_i(w)$, which however may be arbitrary.

Proposition. *Let $n > 2f$. If Assumptions 3 and 4 hold, then for all $w \in \mathbb{R}^d$ and $j \in [d]$*

$$\mathbb{E}\left\|\text{TR}\left(\{G_1(w)\}_j, \ldots, \{G_n(w)\}_j\right) - \{\nabla \mathcal{L}(w)\}_j\right\|^2 \leq \Delta(n,f)\frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}}\sigma_i^2$$

where $\Delta(n,f) = \frac{2|\mathcal{C}|^2(t+1)}{(|\mathcal{C}|-t)^2}$ and $\sigma_i^2 = \left(\frac{M_w}{s_i} + \delta_{i,j}^2\right) + \frac{Q_w}{s_i}\{\nabla \mathcal{L}(w)\}_j^2$.

Proof. We consider an arbitrary w and j. We denote $\{G_i(w)\}_j$, $\text{TR}\left(\{G_1(w)\}_j, \ldots, \{G_n(w)\}_j\right)$ and $\{\nabla \mathcal{L}(w)\}_j$ simply by G_{ij}, TM_j and $\nabla_j \mathcal{L}$, respectively. From a prior result [27, proof of Theorem 1], we have

$$(\text{TM}_j - \nabla_j \mathcal{L})^2 \leq \frac{2}{(|\mathcal{C}|-t)^2}\left(\left(\sum_{i\in\mathcal{C}}(G_{ij} - \nabla_j \mathcal{L})\right)^2 + t\sum_{i\in\mathcal{C}}(G_{ij} - \nabla_j \mathcal{L})^2\right). \tag{7}$$

Using the above, we obtain below an upper bound on $\mathbb{E}_{\zeta_i}\left[(G_{ij} - \nabla_j \mathcal{L})^2\right]$ for an arbitrary $i \in \mathcal{C}$. Note that

$$\begin{aligned}\mathbb{E}_{\zeta_i}\left[(G_{ij} - \nabla_j \mathcal{L})^2\right] &= \mathbb{E}_{\zeta_i}\left[(G_{ij} - \mathbb{E}_{\zeta_i}[G_{ij}] + \mathbb{E}_{\zeta_i}[G_{ij}] - \nabla_j \mathcal{L})^2\right] \\ &= \mathbb{E}_{\zeta_i}\left[(G_{ij} - \mathbb{E}_{\zeta_i}[G_{ij}])^2\right] + (\mathbb{E}_{\zeta_i}[G_{ij}] - \nabla_j \mathcal{L}])^2 \end{aligned} \tag{8}$$

where recall that ζ_i denotes the mini-batch (of size s_i) chosen randomly by worker i from its local dataset S_i. Note that $\mathbb{E}_{\zeta_i}[G_{ij}] = \frac{1}{|S_i|}\sum_{x\in S_i}[\nabla l(w, x)]_j$. Thus, under Assumption 4,

$$\mathbb{E}_{\zeta_i}\left[(G_{ij} - \mathbb{E}_{\zeta_i}[G_{ij}])^2\right] = \frac{1}{s_i}\mathbb{E}_{x\sim S_i}\left[(\{\nabla l(w, x)\}_j - \mathbb{E}_{p\sim S_i}[\{\nabla l(w, x)\}_j])^2\right] \leq \frac{1}{s_i}\left(M_w + Q_w(\nabla_j \mathcal{L})^2\right).$$

Substituting from above in (8) and using the fact that $(\mathbb{E}_{\zeta_i}[G_{ij}] - \nabla_j \mathcal{L})^2 < \delta_{i,j}^2$, we obtain that

$$\mathbb{E}_{\zeta_i}\left[(G_{ij} - \nabla_j \mathcal{L})^2\right] \leq \frac{M_w}{s_i} + \delta_{i,j}^2 + \frac{Q_w}{s_i}(\nabla_j \mathcal{L})^2. \tag{9}$$

From Jensen's inequality and the fact that correct workers sample mini-batches independently, we have

$$\mathbb{E}\left[\left(\sum_{i\in\mathcal{C}}(G_{ij} - \nabla_j \mathcal{L})\right)^2\right] \leq |\mathcal{C}|\sum_{i\in\mathcal{C}}\mathbb{E}[G_{ij} - \nabla_j \mathcal{L}]^2 = |\mathcal{C}|\sum_{i\in\mathcal{C}}\mathbb{E}_{\zeta_i}[G_{ij} - \nabla_j \mathcal{L}]^2.$$

Substituting above from (9) we obtain that

$$\mathbb{E}\left[\left(\sum_{i\in\mathcal{C}}(G_{ij} - \nabla_j \mathcal{L})\right)^2\right] \leq |\mathcal{C}|\left(\sum_{i\in\mathcal{C}}\left(\frac{M_w}{s_i} + \delta_{i,j}^2\right) + \left(\sum_{i\in\mathcal{C}}\frac{Q_w}{s_i}\right)(\nabla_j \mathcal{L})^2\right). \tag{10}$$

Upon taking the expectation on both sides in (7), and then substituting from (9) and (10) we obtain that

$$\begin{aligned}\mathbb{E}\left[(\text{TM}_j - \nabla_j \mathcal{L})^2\right] &\leq \frac{2}{(|\mathcal{C}|-t)^2}\left(\mathbb{E}\left[\left(\sum_{i\in\mathcal{C}}(G_{ij} - \nabla_j \mathcal{L})\right)^2\right] + t\sum_{i\in\mathcal{C}}\mathbb{E}_{\zeta_i}\left[(G_{ij} - \nabla_j \mathcal{L})^2\right]\right) \\ &\leq \frac{2|\mathcal{C}|(t+1)}{(|\mathcal{C}|-t)^2}\left(\sum_{i\in\mathcal{C}}\left(\frac{M_w}{s_i} + \delta_{i,j}^2\right) + \left(\sum_{i\in\mathcal{C}}\frac{Q_w}{s_i}\right)(\nabla_j \mathcal{L})^2\right)\end{aligned}$$

The above proofs the proposition. □

From Proposition **??**, we obtain that for a coordinate-wise trimmed mean, we get:

$$\mathbb{E}\left\|\mathrm{TM}-[\nabla\mathcal{L}]_{\mathcal{B}}\right\|^2 \leq \mathbb{E}\left[\sum_{j\in[b]}\left(\mathrm{TM}_j-\nabla_j\mathcal{L}\right)^2\right] \tag{11}$$

$$\leq \sum_{j\in[b]}\left(\frac{2|\mathcal{C}|(t+1)}{(|\mathcal{C}|-t)^2}\left(\sum_{i\in\mathcal{C}}\left(\frac{M}{s_i}+\delta_{i,j}^2\right)+\left(\sum_{i\in\mathcal{C}}\frac{Q}{s_i}\right)\nabla_j\mathcal{L}^2\right)\right) \tag{12}$$

$$\leq \underbrace{\frac{2|\mathcal{C}|^2(t+1)}{(|\mathcal{C}|-t)^2}}_{\Delta}\frac{1}{\mathcal{C}}\sum_{i\in\mathcal{C}}\underbrace{\left(b\left(\frac{M}{s_i}+\delta_i^2\right)+\frac{Q}{s_i}\left\|[\nabla\mathcal{L}]_{\mathcal{B}}\right\|^2\right)}_{\sigma_i^2} \text{ where } (\delta_i=\max_{j\in[b]}\delta_{i,j}) \tag{13}$$

B.2 Proof of Lemma 1

To prove Lemma 1 we first show the following intermediate result.

Lemma 2. *Suppose that Assumptions 3 and 4 hold true. Consider the k-th step of Algorithm 1. If* AR *is $\Delta(v,f)$ robust at w_k then*

$$\mathbb{E}\left[\left\|AG(w_k)-[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right] \leq \Delta(v,f)\left(A_{w_k}+B_{w_k}\left\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right) \tag{14}$$

where $A_{w_k}=\sum_{j\in\mathcal{B}_k}\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\left(\frac{M_{w_k}}{s_i'}+\delta_{i,j}^2\right)\right)$, $B_w=\sqrt{\sum_{j\in\mathcal{B}_k}\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\frac{Q_{w_k}}{s_i'}\right)^2}$, *and $\mathcal{C}_j$ denotes the set of honest workers that computed coordinate j.*

Proof. For convenience, we write w_k simply as w. First, we consider an arbitrary correct worker i and block $\mathcal{B}\subseteq[d]$. Recall that $G_i(w)=\frac{1}{s_i}\sum_{x\in\zeta_i}\nabla l(w,x)$ where ζ_i is a mini-batch of s_i data points sampled randomly from $\mathcal{S}_i$. Note that

$$\mathbb{E}_{\zeta_i}\left[\left\|[G_i(w)]_{\mathcal{B}}-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2\right]=\underbrace{\mathbb{E}_{\zeta_i}\left[\left\|[G_i(w)]_{\mathcal{B}}-\mathbb{E}_{\zeta_i}\left[[G_i(w)]_{\mathcal{B}}\right]\right\|^2\right]}_{\mathbf{A}}+\underbrace{\left\|\mathbb{E}_{\zeta_i}\left[[G_i(w)]_{\mathcal{B}}\right]-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2}_{\mathbf{B}}. \tag{15}$$

We obtain below upper bounds on **A** and **B**. In both cases, we use the fact that $\mathbb{E}_{\zeta_i}\left[[G_i(w)]_{\mathcal{B}}\right]=\mathbb{E}_{x\sim\mathcal{S}_i}\left[[\nabla l(w,x)]_{\mathcal{B}}\right]$ where notation $x\sim\mathcal{S}_i$ denotes that each data point in $\mathcal{S}_i$ has uniform probability.

A). Thus, As ζ_i is a mini-batch obtained by sampling s_i random data points from $\mathcal{S}_i$ without replacements [20],

$$\mathbb{E}_{\zeta_i}\left[\left\|[G_i(w)]_{\mathcal{B}}-\mathbb{E}_{\zeta_i}\left[[G_i(w)]_{\mathcal{B}}\right]\right\|^2\right]=\mathbb{E}_{z\sim\mathcal{S}_i}\left[\left\|[\nabla l(w,z)]_{\mathcal{B}}-\mathbb{E}_{x\sim\mathcal{S}_i}\left[[\nabla l(w,x)]_{\mathcal{B}}\right]\right\|^2\right]\left(\frac{|\mathcal{S}_i|-s_i}{s_i(|S_i|-1)}\right)$$

where $s_i=|\zeta_i|$. By definition of $[\cdot]_{\mathcal{B}}$, $\forall y\in\mathbb{R}^d$ we have $\left\|[y]_{\mathcal{B}}\right\|^2=\sum_{j\in\mathcal{B}}\left(\{y\}_j\right)^2$. Hence, from above

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

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

$$\mathbb{E}_{\zeta_i}\left[\left\|[G_i(w)]_{\mathcal{B}}-\mathbb{E}_{\zeta_i}\left[[G_i(w)]_{\mathcal{B}}\right]\right\|^2\right]\leq\sum_{j\in\mathcal{B}}\left(\frac{M_w}{s_i'}+\frac{Q_w}{s_i'}\left(\{\nabla\mathcal{L}(w)\}_j\right)^2\right)=\frac{bM_w}{s_i'}+\frac{Q_w}{s_i'}\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2 \tag{16}$$

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

B). Similarly,

$$\left\|\mathbb{E}_{\zeta_i}\left[[G_i(w)]_{\mathcal{B}}\right]-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2=\left\|\mathbb{E}_{x\sim\mathcal{S}_i}\left[[\nabla l(w,x)]_{\mathcal{B}}\right]-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2=\sum_{j\in\mathcal{B}}\left\|\mathbb{E}_{p\sim S_i}\left[[\nabla l_p(w)]_j\right]-[\nabla\mathcal{L}(w)]_j\right\|^2.$$

Substituting from Assumption 3 above, we obtain that

$$\left\|\mathbb{E}_{\zeta_i}\left[[G_i(w)]_{\mathcal{B}}\right]-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2\leq\sum_{j\in\mathcal{B}}\delta_{i,j}^2. \tag{17}$$

A+B). Substituting from (16) and (17) in (15) we obtain that

$$\mathbb{E}_{\zeta_i}\left[\left\|[G_i(w)]_{\mathcal{B}}-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2\right]\leq\frac{bM_w}{s_i'}+\sum_{j\in\mathcal{B}}\delta_{i,j}^2+\frac{Q_w}{s_i'}\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2. \tag{18}$$

As $\mathcal{B}$ is an arbitrary subset of $[d]$, (18) implies that for all $j\in\mathcal{B}$,

$$\mathbb{E}_{\zeta_i}\left[(\{G_i(w)\}_j-\{\nabla\mathcal{L}(w)\}_j)^2\right]\leq\left(\frac{M_w}{s_i'}+\delta_{i,j}^2\right)+\frac{Q_w}{s_i'}\left(\{\nabla\mathcal{L}(w)\}_j\right)^2. \tag{19}$$

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 (18) holds for an arbitrary correct worker $i\in\mathcal{C}$, we obtain that

$$\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\mathbb{E}_{\zeta_i}\left[(\{G_i(w)\}_j-\{\nabla\mathcal{L}(w)\}_j)^2\right]\leq\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\left(\frac{M_w}{s_i'}+\delta_{i,j}^2\right)+\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\frac{Q_w}{s_i'}\{\nabla\mathcal{L}(w)\}_j^2. \tag{20}$$

Now, as AR is assumed $\Delta(v,f)$-robust at w, by Definition 1,

$$\mathbb{E}(\{AG(w)\}_j-\{\nabla\mathcal{L}(w)\}_j)^2\leq\Delta(v,f)\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\mathbb{E}_{\zeta_i}\left[(\{G_i(w)\}_j-\{\nabla\mathcal{L}(w)\}_j)^2\right]\right)$$

Substituting from (20) above we obtain that

$$\mathbb{E}(\{AG(w)\}_j-\{\nabla\mathcal{L}(w)\}_j)^2\leq\Delta(v,f)\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\left(\frac{M_w}{s_i'}+\delta_{i,j}^2\right)+\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\left(\frac{Q_w}{s_i'}\{\nabla\mathcal{L}(w)\}_j^2\right)\right) \tag{21}$$

By summing both sides in (21) over all $j\in\mathcal{B}_k$ we obtain that

$$\begin{aligned}&\mathbb{E}\left[\left\|[AG(w)]_{\mathcal{B}_k}-[\nabla\mathcal{L}(w)]_{\mathcal{B}_k}\right\|^2\right]\leq\\&\Delta(v,f)\left(\sum_{j\in\mathcal{B}_k}\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\left(\frac{M_w}{s_i'}+\delta_{i,j}^2\right)\right)+\sum_{j\in\mathcal{B}_k}\left(\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\frac{Q_w}{s_i'}\right)\{\nabla\mathcal{L}(w)\}_j^2\right)\right)\end{aligned} \tag{22}$$

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

$$\sum_{j\in\mathcal{B}_k}\left(\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\frac{Q_w}{s_i'}\right)\{\nabla\mathcal{L}(w)\}_j^2\right)\leq\left(\sum_{j\in\mathcal{B}_k}\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\frac{Q_w}{s_i'}\right)^2\right)^{\frac{1}{2}}\left(\sum_{j\in\mathcal{B}_k}\{\nabla\mathcal{L}(w)\}_j^4\right)^{\frac{1}{2}} \tag{23}$$

Recall that $\forall a_i>0$, $\left(\sum_i a_i^2\right)^{\frac{1}{2}}\leq\sum_i a_i$. Thus,

$$\left(\sum_{j\in\mathcal{B}_k}\{\nabla\mathcal{L}(w)\}_j^4\right)^{\frac{1}{2}}\leq\sum_{j\in\mathcal{B}_k}\{\nabla\mathcal{L}(w)\}_j^2 \tag{24}$$

Recall that $[AG(w)]_{\mathcal{B}_k} = AG(w)$. Hence, substituting from (23) and (24) in (22) concludes the proof, i.e., we have

$$\mathbb{E}\left[\left\|AG(w) - [\nabla \mathcal{L}(w)]_{\mathcal{B}}\right\|^2\right] \leq$$
$$\Delta\left(\sum_{j\in\mathcal{B}_k}\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\left(\frac{M_w}{s_i'}+\delta_{i,j}^2\right)\right)+\left(\sum_{j\in\mathcal{B}_k}\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\frac{Q_w}{s_i'}\right)^2\right)^{\frac{1}{2}}\left(\sum_{j\in\mathcal{B}_k}\{\nabla\mathcal{L}(w)\}_j^2\right)\right).$$

□

We are now ready to prove Lemma 1, which is stated again below for convenience.

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

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

Proof. For convenience, we drop the iteration index k in the proof. From Lemma 2 we have

$$\underbrace{\Delta(v,f)\left(A_w + B_w\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2\right)}_{:=r^2} \geq \mathbb{E}\left[\left\|AG(w)-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2\right] \geq \left\|\mathbb{E}[AG(w)]-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2 \tag{25}$$

where the second inequality follows from Jensen's inequality. The fact that $\left\|\mathbb{E}[AG(w)]-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2 \leq r^2$ means that $\mathbb{E}[AG(w)]$ belongs to a ball centered at $[\nabla\mathcal{L}(w_k)]_{\mathcal{B}}$ with radius r, illustrated in Figure 9 below.

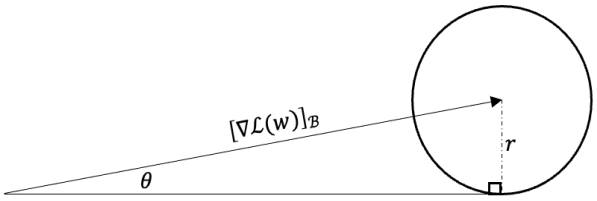

Figure 9: When $\left\|\mathbb{E}\left[AG(w)\right]-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\| \leq r$, $\mathbb{E}[AG(w)]$ belongs to a ball centered at $[\nabla\mathcal{L}(w)]_{\mathcal{B}}$ with radius r.

From (25) we obtain that

$$\left\|\mathbb{E}[AG(w)]-[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2 = \left\|\mathbb{E}[AG(w)]\right\|^2 + 2\langle\mathbb{E}[AG(w)], [\nabla\mathcal{L}(w)]_{\mathcal{B}}\rangle + \left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2 \leq r^2$$

Thus,

$$2\left\langle\mathbb{E}[AG(w)],\, [\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\rangle \geq \left\|\mathbb{E}[AG(w)]\right\|^2 + \left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2 - r^2. \tag{26}$$

As we assume that $w\in\mathcal{W}$, $\quad\Delta(v,f)A_w \leq (1-\Delta B_w)\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2$. Thus,

$$\frac{r^2}{\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2} = \frac{\Delta A_w + \Delta B_w\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2}{\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2} \leq 1, \text{ which implies that } \frac{r}{\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|} \leq 1. \tag{27}$$

Owing to (27), we consider $\theta\in[0,\,\pi/2]$ such that $\sin\theta = \frac{r}{\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|}$. From triangle inequality, $\left\|\mathbb{E}[AG(w)]\right\| \geq \left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\| - r$. Therefore, $\left\|\mathbb{E}[AG(w)]\right\| \geq (1-\sin\theta)\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\| \geq 0$. Using these in (26) we obtain that

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

Thus,

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

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

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

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

$$\langle \mathbb{E}[AG(w)], [\nabla \mathcal{L}(w)]_{\mathcal{B}} \rangle \geq \frac{1}{2} \left((1 - B_w \Delta(v, f)) \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 - \Delta(v, f) A_w \right).$$

□

B.3 Proof of Theorem 1

Theorem (Strongly convex convergence)**.** *Suppose that Assumptions 1, 2, 3 and 4 hold true. Consider Algorithm 1 with a constant learning rate* $\gamma < \min_{k \in [T]} \left\{ \frac{1}{L_{\mathcal{B}_k}} \right\}$. *If* $\Delta B_{w_k} < 1$ *then*

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

where $\mathcal{L}^*$ *denotes the minimum value of* $\mathcal{L}(w)$, $b = \min_{k \in [T]} |\mathcal{B}_k|$, $A = \max_{k \in [T]} A_{w_k}$, $B = \max_{k \in [T]} B_{w_k}$, *and* $H = \frac{d\Delta A}{2b\mu(1 - \Delta B)}$.

Proof. Recall the definition of set $\mathcal{W}$ from (5), i.e.,

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

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 $w_s \in \mathcal{W}$.

A). All the elements of the sequence $\{w_k\}_{k>0}$ are in the subspace $\mathcal{W}$. With $\gamma_k = \gamma$ for all k, $w_{k+1} = w_k - \gamma AG(w_k)$. Now, we consider an arbitrary $k \in [T-1]$. Under Assumption 1, i.e., component-wise Lipschitz continuous of $\nabla \mathcal{L}(w)$ with coefficient L, and the fact that $AG(w_k) = [AG(w_k)]_{\mathcal{B}_k}$ we have [22, Section 2.1]]

$$\mathcal{L}(w_{k+1}) = \mathcal{L}(w_k - \gamma_k AG(w_k)) \leq \mathcal{L}(w_k) - \gamma_k \langle [\nabla F_k(w_k)]_{\mathcal{B}_k}, AG(w_k) \rangle + \gamma_k^2 \frac{L_{\mathcal{B}_k}}{2} \left\| AG(w_k) \right\|^2 \tag{28}$$

where $L_{\mathcal{B}_k} = \sum_{j \in \mathcal{B}_k} L_j$ is the Lipschitz coefficient of the truncated gradient $[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}$. Using the fact that: $\left\| [\nabla F_k]_{\mathcal{B}_k} - AG(w_k) \right\|^2 = \left\| [\nabla F_k]_{\mathcal{B}_k} \right\|^2 - 2\langle [\nabla F_k]_{\mathcal{B}_k}, AG(w_k) \rangle + \left\| AG(w_k) \right\|^2$, Inequality (28) becomes:

$$\begin{aligned} \mathcal{L}(w_{k+1}) \leq \mathcal{L}(w_k) - \gamma_k (1 - \gamma_k L_{\mathcal{B}_k}) \langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, AG(w_k) \rangle \\ + \gamma_k^2 \frac{L_{\mathcal{B}_k}}{2} \left(\left\| AG(w_k) - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \right\|^2 - \left\| [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \right\|^2 \right) \end{aligned} \tag{29}$$

Introducing the expectation $\mathbb{E}_1$ over the randomness induced by the mini-batch selection of honest workers and the behavior of Byzantine workers:

$$\begin{aligned} \mathbb{E}_1[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \gamma_k (1 - \gamma_k L_{\mathcal{B}_k}) \langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, \mathbb{E}[AG(w_k)] \rangle \\ + \gamma_k^2 \frac{L_{\mathcal{B}_k}}{2} \left(\mathbb{E} \left\| AG(w_k) - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \right\|^2 - \left\| [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \right\|^2 \right) \end{aligned} \tag{30}$$

Using Lemma 1 and the fact that $\gamma_k < \frac{1}{L_{\mathcal{B}_k}}$ we obtain:

$$\begin{aligned} \mathbb{E}_1[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{1}{2} \gamma_k (1 - \gamma_k L_{\mathcal{B}_k}) ((1 - \Delta B_{w_k}) \left\| [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \right\|^2 - \Delta A_{w_k}) \\ + \gamma_k^2 \frac{L_{\mathcal{B}_k}}{2} \left(\mathbb{E} \left\| AG(w_k) - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \right\|^2 - \left\| [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \right\|^2 \right) \end{aligned} \tag{31}$$

Using Lemma 2 in (31), we also have:

$$\begin{aligned}\mathbb{E}_1[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{1}{2}\gamma_k(1-\gamma_k L_{\mathcal{B}_k})((1-\Delta B_{w_k})\left\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2 - \Delta A_{w_k}) \\ +\gamma_k^2 \frac{L_{\mathcal{B}_k}}{2}\left(\Delta\left(A_{w_k} + B_{w_k}\left\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right) - \left\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right)\end{aligned} \tag{32}$$

Rearranging the terms gives:

$$\mathbb{E}_1[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\gamma_k}{2}(1-\Delta B_{w_k})\left\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2 + \frac{\gamma_k \Delta A_{w_k}}{2} \tag{33}$$

Now, introducing the double expectation $\mathbb{E}_2 = \mathbb{E}_1\,\mathbb{E}_{\mathcal{B}}$ where $\mathbb{E}_{\mathcal{B}}$ is the expectation taken over the random block $\mathcal{B}_k$:

$$\mathbb{E}_2[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\gamma_k}{2}(1-\Delta B_{w_k})\,\mathbb{E}_{\mathcal{B}}\left\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2 + \frac{\gamma_k \Delta A_{w_k}}{2} \tag{34}$$

$$\leq \mathcal{L}(w_k) - \frac{b_k}{d}\frac{\gamma_k}{2}(1-\Delta B_{w_k})\left\|\nabla \mathcal{L}(w_k)\right\|^2 + \frac{\gamma_k \Delta A_{w_k}}{2} \tag{35}$$

Using the Polyak-Lojasiewicz inequality in (35) gives:

$$\mathbb{E}_2[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{b_k}{d}\mu\gamma_k(1-\Delta B_{w_k})(\mathcal{L}(w_k) - \mathcal{L}^*) + \frac{\gamma_k \Delta A_{w_k}}{2} \tag{36}$$

If we now subtract $H_k = \frac{d\Delta A_{w_k}}{2b_k\mu(1-\Delta B_{w_k})}$ and $\mathcal{L}^*$ from both sides of (36), we obtain:

$$\mathbb{E}_2[\mathcal{L}(w_{k+1})] - \mathcal{L}^* - H_k \leq \left(1 - \frac{b_k}{d}\mu\gamma_k(1-\Delta B_{w_k})\right)(\mathcal{L}(w_k) - \mathcal{L}^* - H_k) \tag{37}$$

Let $b = \min_{k\in[T]} b_k$, $A = \max_{k\in[T]} A_{w_k}$, $B = \max_{k\in[T]} B_{w_k}$ and $\gamma = \min_{k\in[T]} \gamma_k$. Using these upper bounds in (37) gives:

$$\mathbb{E}_2[\mathcal{L}(w_{k+1})] - \mathcal{L}^* - H \leq \left(1 - \frac{b}{d}\mu\gamma(1-\Delta B)\right)(\mathcal{L}(w_k) - \mathcal{L}^* - H) \tag{38}$$

Let $\mathbb{E}$ denote the total expectation taken over all the randomness generated in Algorithm 1 through iterations $(1, \ldots, T)$. Introducing this expectation on (39) and subtracting $\mathcal{L}^*$ from both sides, we get:

$$\mathbb{E}[\mathcal{L}(w_{k+1})] - \mathcal{L}^* - H \leq \left(1 - \frac{b}{d}\mu\gamma(1-\Delta B)\right)(\mathbb{E}[\mathcal{L}(w_k)] - \mathcal{L}^* - H) \tag{39}$$

By assumption, $\Delta B < 1$. We also know that $\mu < L_{\mathcal{B}_k}$, which means that: $0 < \frac{b}{d}\mu\gamma(1-\Delta B) < 1$. Inequality (39) is therefore a contraction inequality. By applying it repeatedly through iterations $k \in [0, \cdots, K-1]$, we obtain:

$$\mathbb{E}\,\mathcal{L}(w_K) - \mathcal{L}^* \leq \left(1 - \frac{b}{d}\mu\gamma(1-\Delta B)\right)^{K-1}(\mathcal{L}(w_1) - \mathcal{L}^* - H) + H \tag{40}$$

B). There exists at least one element w_s of the sequence $\{w_k\}_{k>0}$ in the subspace $\mathbb{R}^d \setminus \mathcal{W}$. By definition of $\mathcal{W}$,

$$\left\|[\nabla \mathcal{L}(w_s)]_{\mathcal{B}_s}\right\|^2 < \frac{\Delta A_{w_s}}{1-\Delta B_{w_s}}. \tag{41}$$

Thus,

$$\mathbb{E}\left\|[\nabla \mathcal{L}(w_s)]_{\mathcal{B}_s}\right\|^2 < \frac{\Delta A_{w_s}}{1-\Delta B_{w_s}} \tag{42}$$

$$\frac{b_s}{d}\left\|\nabla \mathcal{L}(w_s)\right\|^2 < \frac{\Delta A_{w_s}}{1-\Delta B_{w_s}} \tag{43}$$

$$\left\|\nabla \mathcal{L}(w_s)\right\|^2 < \frac{d\Delta A_{w_s}}{b_s(1-\Delta B_{w_s})} \tag{44}$$

Using the Polyak-Lojasiewicz inequality, we have

$$\|\nabla \mathcal{L}(w_s)\|^2 \geq 2\mu(\mathcal{L}(w_s) - \mathcal{L}^*) \tag{45}$$

Combining (44) and (45) gives:

$$\mathcal{L}(w_s) - \mathcal{L}^* \leq \underbrace{\frac{d\Delta A_{w_s}}{2b_s\mu(1-\Delta B_{w_s})}}_{H_s} \tag{46}$$

A+B). From (46) and (40) we obtain, $\forall w \in \mathbb{R}^d$:

$$\mathbb{E}\,\mathcal{L}(w_K) - \mathcal{L}^* \leq \max\left\{\left(1 - \frac{b}{d}\mu\gamma(1-\Delta B)\right)^{K-1}(\mathcal{L}(w_1) - \mathcal{L}^* - H) + H, H_s\right\} \tag{47}$$

$$= \left(1 - \frac{b}{d}\mu\gamma(1-\Delta B)\right)^{K-1}(\mathcal{L}(w_1) - \mathcal{L}^* - H) + H \quad (\text{Since } H_s < H) \tag{48}$$

which concludes the proof. □

B.4 Proof of Theorem 2

Theorem (Non-convex convergence)**.** *Suppose that Assumptions 1, 3 and 4 hold. If $\Delta B_k < 1$ and $\gamma_k < \frac{1}{L_{\mathcal{B}_k}}$, then we have:*

$$\min_{1<i<T} \mathbb{E}\left[\|\nabla \mathcal{L}(w_i)\|^2\right] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{Tb\gamma(1-\Delta B)} + \frac{d\Delta A}{b(1-\Delta B)}$$

where $A = \max_{k\in[T]} A_{w_k}$, $B = \max_{k\in[T]} B_{w_k}$, $\gamma = \min_{k\in[T]} \gamma_k$ and $b = \min_{k\in[T]} b_k$.

Proof. As in the proof of the previous theorem, we suppose that the server constructs a consistent block $\mathcal{B}_k$ of size b_k, at iteration k and we decompose this proof into two parts:

A). All the elements of the sequence $\{w_k\}_{k>0}$ are in the subspace $\mathcal{W}$.

Let us recall (35):

$$\mathbb{E}_2[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{b_k}{2d}\gamma_k(1-\Delta B_{w_k})\|\nabla \mathcal{L}(w_k)\|^2 + \frac{\gamma_k \Delta A_{w_k}}{2} \tag{49}$$

Let $A = \max_{k\in[T]} A_{w_k}$, $B = \max_{k\in[T]} B_{w_k}$, $\gamma = \min_{k\in[T]} \gamma_k$ and $b = \min_{k\in[T]} b_k$. Using these upper bounds, we obtain:

$$\mathbb{E}_2[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{b\gamma}{2d}(1-\Delta B)\|\nabla \mathcal{L}(w_k)\|^2 + \frac{\gamma\Delta A}{2} \tag{50}$$

Rearranging the terms gives:

$$\|\nabla \mathcal{L}(w_k)\|^2 \leq \frac{2d}{b\gamma}(\mathcal{L}(w_k) - \mathbb{E}_2[\mathcal{L}(w_{k+1})]) + \frac{d\Delta A}{b(1-\Delta B)} \tag{51}$$

Let $\mathbb{E}$ denote the total expectation taken over all the randomness generated in Algorithm 1 through iterations $(1, \ldots, T)$. Introducing this expectation on the last inequality, we get:

$$\mathbb{E}\|\nabla \mathcal{L}(w_k)\|^2 \leq \frac{2d}{b\gamma(1-\Delta B)}(\mathbb{E}[\mathcal{L}(w_k)] - \mathbb{E}[\mathcal{L}(w_{k+1})]) + \frac{d\Delta A}{b(1-\Delta B)} \tag{52}$$

Summing both sides through $[1, \cdots, T]$ and

$$\sum_{i=A}^{T} \mathbb{E}\|\nabla \mathcal{L}(w_i)\|^2 \leq \frac{2d}{b\gamma(1-\Delta B)}(\mathbb{E}[\mathcal{L}(w_1)] - \mathbb{E}[\mathcal{L}(w_T)]) + T\frac{d\Delta A}{b(1-\Delta B)} \tag{53}$$

Dividing the last inequality by T gives:

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}\left\|\nabla\mathcal{L}(w_i)\right\|^2\right] \leq \frac{2d}{Tb\gamma(1-\Delta B)}(\mathcal{L}(w_1)-\mathbb{E}[\mathcal{L}(w_T)]) + \frac{d\Delta A}{b(1-\Delta B)} \tag{54}$$

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

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}\left\|\nabla\mathcal{L}(w_i)\right\|^2\right] \leq \frac{2d}{Tb\gamma(1-\Delta B)}(\mathcal{L}(w_1)-\mathcal{L}(w_*)) + \frac{d\Delta A}{b(1-\Delta B)} \tag{55}$$

which also means that:

$$\min_{1<i<T}\mathbb{E}\left[\left\|\nabla\mathcal{L}(w_i)\right\|^2\right] \leq \frac{2d}{kb\gamma(1-\Delta B)}(\mathcal{L}(w_1)-\mathcal{L}(w_*)) + \frac{d\Delta A}{b(1-\Delta B)} \tag{56}$$

B). There exists at least one element w_s of the sequence $\{w_k\}_{k>0}$ in the subspace $\mathbb{R}^d \setminus \mathcal{W}$.

By construction of $\mathcal{W}$, as in (44), we have:

$$\left\|\nabla\mathcal{L}(w_s)\right\|^2 < \frac{d\Delta A_{w_s}}{b_s(1-\Delta B_{w_s})} \tag{57}$$

A+B). By combining (55) and (57), we obtain, $\forall w \in \mathbb{R}^d$:

which also means that:

$$\min_{1<i<T}\mathbb{E}\left[\left\|\nabla\mathcal{L}(w_i)\right\|^2\right] \leq \max\left\{\frac{2d(\mathcal{L}(w_1)-\mathcal{L}(w_*))}{kb\gamma(1-\Delta B)} + \frac{2d\Delta A}{b(1-\Delta B)}, \frac{d\Delta A_{w_s}}{b_s(1-\Delta B_{w_s})}\right\} \tag{58}$$

$$= \frac{2d(\mathcal{L}(w_1)-\mathcal{L}(w_*))}{kb\gamma(1-\Delta B)} + \frac{d\Delta A}{b(1-\Delta B)} \tag{59}$$

and that concludes the proof. □

C 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 HGO against SGD. We also use the datasets CIFAR-10, MNIST, BOSTON and PHISHING. In the following, we describe their properties as well as the pre-processing steps adopted for each one of them.

C.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:

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

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 10. 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): $ReLU(z) = \max(0, z)$

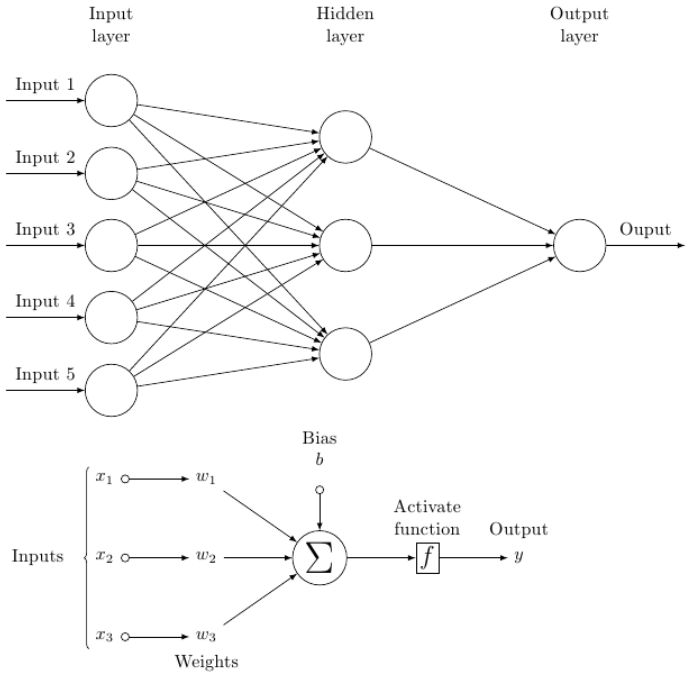

Figure 10: A feed-forward neural network with a 5-inputs layer, one hidden layer with 3 neurons and one output layer with only one class. Each neuron is composed of a transfer function summing all the inputs and a bias, followed by an activation function.

C.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.

CIFAR-10. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. 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.

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.

C.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.

C.4 Attacks used in the Byzantine experiments

It is known that AVERAGE is not a robust aggregation rule. In fact, as shown in [4], 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 HGO 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 HGO 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 [3]. 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 2 summarizes the attack.

Algorithm 2 "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 (μ_j) and standard deviation (σ_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 [28]. 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 ϵ. Formally, if $\mathcal{C}$ is the set of correct workers, then:

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

C.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.

C.6 Source code

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

- Distributed implementation:
  `https://anonymous.4open.science/r/HgO-5136`
- Android application:
  `https://anonymous.4open.science/r/HgOApp-CC8E`

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

D HgO: Android application

In the previous section, we implemented HGO 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 HGO 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.

D.1 Setup

The smartphone implementation of HGO uses the python cross-platform library Kivy (`https://kivy.org`) and is optimized to deploy the Android version[4] of HGO. 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.

[4]The application is cross-platform and can also be deployed on IOS, Linux, MacOS and Windows.

D.2 Tutorials

To join the distributed training from the Android application, the application first needs to be configured (Figure **??**, 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.

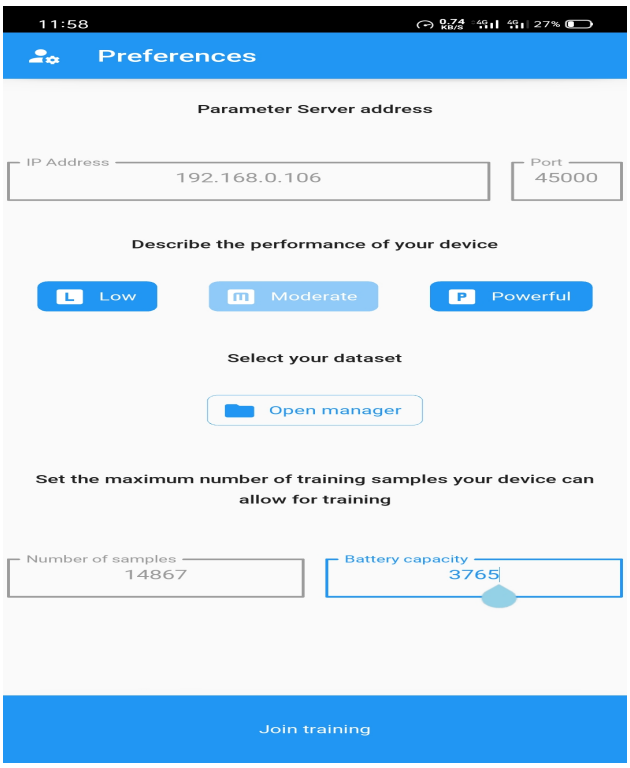

Figure 11: HGO Configuration screen on an Android smartphone.

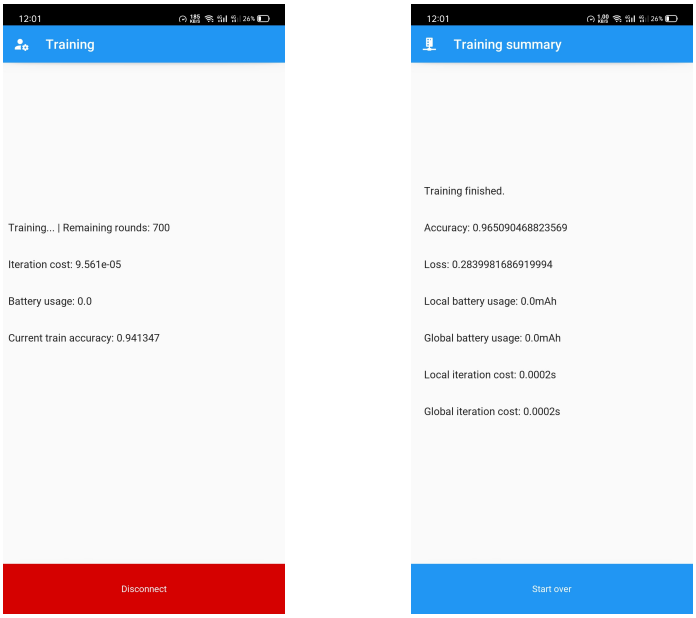

Figure 12: HGO running on an Android smartphone. The left side figure shows the app status during the training, and the right side figure shows the final screen, when the training is complete.