Efficient Detection and Filtering Systems for Distributed Training

Konstantinos Konstantinidis and Aditya Ramamoorthy, *Senior Member, IEEE*

***Abstract*—A plethora of modern machine learning tasks requires the utilization of large-scale distributed clusters as a critical component of the training pipeline. However, abnormal Byzantine behavior of the worker nodes can derail the training and compromise the quality of the inference. Such behavior can be attributed to unintentional system malfunctions or orchestrated attacks; as a result, some nodes may return arbitrary results to the parameter server (PS) that coordinates the training. Recent work considers a wide range of attack models and has explored robust aggregation and/or computational redundancy to correct the distorted gradients.**

In this work, we consider attack models ranging from strong ones: q omniscient adversaries with full knowledge of the defense protocol that can change from iteration to iteration to weak ones: q randomly chosen adversaries with limited collusion abilities that only change every few iterations at a time. Our algorithms rely on redundant task assignments coupled with detection of adversarial behavior. For strong attacks, we demonstrate a reduction in the fraction of distorted gradients ranging from 16%-99% as compared to the prior state-of-the-art. Our top-1 classification accuracy results on the CIFAR-10 data set demonstrate a 25% advantage in accuracy (averaged over strong and weak scenarios) under the most sophisticated attacks compared to state-of-the-art methods.

***Index Terms*—Byzantine resilience, distributed training, gradient descent, deep learning, optimization, security.**

I. Introduction and Background

INcreasingly complex machine learning models with large data set sizes are nowadays routinely trained on distributed clusters. A typical setup consists of a single central machine (*parameter server* or PS) and multiple worker machines. The PS owns the data set, assigns gradient tasks to workers, and coordinates the protocol. The workers then compute gradients of the loss function with respect to the model parameters. These computations are returned to the PS, which *aggregates* them, updates the model and maintains the global copy of it. The new copy is communicated back to the workers. Multiple iterations of this process are performed until convergence has been achieved. PyTorch [1], TensorFlow [2], MXNet [3], CNTK [4] and other frameworks support this architecture.

These setups offer significant speedup benefits and enable training of challenging, large-scale models. Nevertheless, they are vulnerable to misbehavior by the worker nodes, i.e., when a subset of them returns erroneous computations to the PS, either inadvertently or on purpose. This "*Byzantine*" behavior can be attributed to a wide range of reasons. The principal causes of inadvertent errors are hardware and software malfunctions (e.g., [5]). Reference [6] exposes the vulnerability of neural networks to such failures and identifies weight parameters that could maximize accuracy degradation. The gradients may also be distorted in an adversarial manner. As ML problems demand more resources, many jobs are often outsourced to external commodity servers (cloud) whose security cannot be guaranteed. Thus, an adversary may be able to gain control of some devices and fool the model. The distorted gradients can derail the optimization and lead to low test accuracy or slow convergence.

Achieving robustness in the presence of Byzantine node behavior and devising training algorithms that can efficiently aggregate the gradients has inspired several works [7]–[18]. Two main ideas deal with the alleviation of Byzantine effects. The first one is concerned with filtering out the corrupted computations from the training without attempting to identify the Byzantine workers. Specifically, many existing papers use majority voting and median-based defenses [7]–[13] for this purpose. In addition, several works also operate by replicating the gradient tasks [14]–[18], allowing for consistency checks across the cluster. The second idea for mitigating Byzantine behavior involves detecting the corrupted devices and subsequently ignoring their calculations [19]–[21], in some instances paired with redundancy [17]. In this work, we propose a technique that combines the usage of redundant tasks, filtering, and detection of Byzantine workers. Our work is applicable to a range of assumptions on the Byzantine behavior.

There is much variability in the adversarial assumptions that prior work considers. For instance, prior work differs in the maximum number of adversaries considered, their ability to collude, their possession of knowledge involving the data assignment and the protocol, and whether the adversarial machines are chosen at random or systematically. For instance, DETOX [16] and DRACO [17] consider adversaries that can only attack a random set of workers. We will initially examine our methods under strong adversarial models similar to those in prior work [10], [11], [14], [22]–[25]. We will then extend our algorithms to tackle weaker failures that are not necessarily adversarial but rather common in commodity machines [5], [6], [26]. We expand on related work in the upcoming Section II.

II. Related Work and Summary of Contributions

A. Related Work

All work in this area (including ours) assumes a reliable parameter server that possesses the global data set and can assign specific subsets of it to workers. *Robust aggregation* methods have also been proposed for federated learning [27], [28]; however, as we make no assumption of privacy, our work,

This work was supported in part by the National Science Foundation (NSF) under grants CCF-1910840 and CCF-2115200. The material in this work has appeared in part at the 2022 IEEE International Symposium on Information Theory (ISIT). (Corresponding author: Aditya Ramamoorthy.)

The authors are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011, USA (e-mail: kostas@iastate.edu; adityar@iastate.edu).

as well as the methods we compare with, do not apply to federated learning.

One category of defenses splits the data set into K *batches* and assigns one to each worker with the ultimate goal of suitably aggregating the results from the workers. Early work in the area [12] established that no *linear aggregation* method (such as averaging) can be robust even to a single adversarial worker. This has inspired alternative methods collectively known as *robust aggregation*. Majority voting, geometric median, and squared-distance-based techniques fall into this category [8]–[13]. While such approaches theoretically allow for robustness against a constant fraction of adversarial nodes (when the number of nodes is large), in practice, they are limited in the number of adversaries that they can deal with when the cluster sizes are up to tens or hundreds of nodes. Moreover, their promised guarantees are often restricted (*e.g.,* only guaranteeing that the output of the aggregator has a positive inner product with the true gradient [12], [24]), which compromises their practicality. In addition, their complexity often scales quadratically with the number of workers [11]. Finally, their theoretical guarantees are insufficient to establish convergence and require strict assumptions such as convexity of the loss function that need to be adjusted for each individual training algorithm.

One of the most popular robust aggregation techniques is known as *mean-around-median* or *trimmed mean* [10], [11], [24]. It handles each dimension of the gradient separately and returns the average of a subset of the values that are closest to the median. *Auror* [25] is a variant of trimmed mean that partitions the values of each dimension into two clusters using *k-means* and discards the smaller cluster if the distance between the two exceeds a threshold; the values of the larger cluster are then averaged. *signSGD* in [26] transmits only the sign of the gradient vectors from the workers to the PS and exploits majority voting to decide the overall update; this practice improves communication time and denies any individual worker too much effect on the update.

Krum in [12] chooses a single honest worker for the next model update, discarding the data from the rest of them. The chosen gradient is the one closest to its $k \in \mathbb{N}$ nearest neighbors. In later work [24], the authors recognized that Krum may converge to an *ineffectual* model in the landscape of non-convex high dimensional problems, such as in neural networks. They showed that a large adversarial change to a single parameter with a minor impact on the L^p norm can make the model ineffective. The same work presents an alternative defense called *Bulyan* to oppose such attacks. The algorithm works in two stages. In the first part, a *selection set* of potentially benign values is iteratively constructed. In the second part, a variant of the trimmed mean is applied to the selection set. Nevertheless, if K machines are used, Bulyan is designed to defend only up to $(K-3)/4$ fraction of corrupted workers. Similarly, the method of [13] is based on the *geometric median of means*.

Another category of defenses is based on *redundancy* and seeks resilience to Byzantines by replicating the gradient computations such that each of them is processed by more than one machine [15]–[18]. Even though this approach requires more computation load, it comes with stronger guarantees of correcting the erroneous gradients. Existing redundancy-based techniques are sometimes combined with robust aggregation [16]. The main drawback of recent work in this category is that the training can be easily disrupted by a powerful, omniscient adversary that has full control of a subset of the nodes and can mount judicious attacks [14].

Redundancy-based method *DRACO* in [17] uses a simple *Fractional Repetition Code* (FRC) (that operates by grouping workers) and the cyclic repetition code introduced in [29], [30] to ensure robustness; majority voting and Fourier decoders try to alleviate the adversarial effects. Their work ensures exact recovery (as if the system had no adversaries) with q Byzantine nodes when each task is replicated $r \geq 2q+1$ times; the bound is information-theoretic minimum, and DRACO is not applicable if it is violated. Nonetheless, this requirement is very restrictive for the typical assumption that up to half of the workers can be Byzantine.

DETOX in [16] extends DRACO and uses a simple grouping strategy to assign the gradients. It performs multiple stages of aggregation to gradually filter the adversarial values. The first stage involves majority voting, while the following stages perform robust aggregation. Unlike DRACO, the authors do not seek exact recovery; hence the minimum requirement in r is small. However, the theoretical resilience guarantees that DETOX provides depend heavily on a "random choice" of the adversarial workers. In fact, we have crafted simple attacks [14] to make this aggregator fail under a more careful choice of adversaries. Furthermore, their theoretical results hold when the fraction of Byzantines is less than $1/40$. Under these assumptions, they show that, on average, a small percentage of the gradient computations will be distorted.

A third category focuses on *ranking* and/or *detection* [17], [19], [20]; the objective is to rank workers using a reputation score to identify suspicious machines and exclude them or give them lower weight in the model update. This is achieved by means of computing reputation scores for each machine or by using ideas from coding theory to assign tasks to workers (encoding) and to detect the adversaries (decoding). *Zeno* in [20] ranks each worker using a score that depends on the estimated loss and the magnitude of the update. Zeno requires strict assumptions on the smoothness of the loss function and the gradient estimates' variance to tolerate an adversarial majority in the cluster. Similarly, *ByGARS* [19] computes reputation scores for the nodes based on an auxiliary data set; these scores are used to weigh the contribution of each gradient to the model update. Its convergence result depends on the assumption of a strongly convex loss function. In contrast, our method does not rely on such restrictive assumptions. Also, we emphasize that these works have not used or constructed worst-case attacks to evaluate the methods in adversarial settings.

B. Contributions

In this paper, we propose novel techniques which combine *redundancy*, *detection*, and *robust aggregation* for Byzantine resilience under a variety of attack modes. We initially present *Aspis*, a subset-based assignment scheme for allocating tasks to workers in strong adversarial settings: omniscient, colluding adversaries, up to q adversaries that can change at each iteration.

In the second part of the paper, we consider weaker attacks: adversaries chosen randomly with limited collusion abilities,

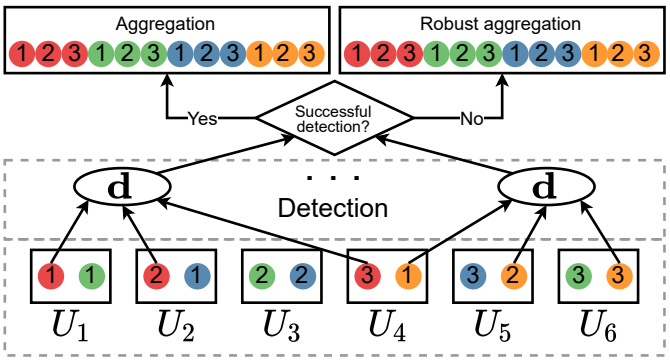


Fig. 1: Aggregation of gradients on a training cluster.

changing only after a few iterations at a time. It is conceivable that Aspis should continue to perform well with weaker attacks. However, as discussed later (Section V-B), Aspis requires large batch sizes (in the mini-batch SGD). It is well-recognized that large batch sizes often cause performance degradation in training [31]. Accordingly, for this class of attacks, we present a different algorithm called *Aspis+* that can work with much smaller batch sizes.

Both Aspis and Aspis+ use combinatorial ideas to assign the tasks to the worker nodes. These allow us to upper bound the fraction of distorted gradients. Our work builds on our initial work in [22] and makes the following contributions.

- Under strong attacks, we propose an attack on Aspis and prove that it is *optimal* in distorting as many files as possible. Even in this adverse scenario, our method enjoys a reduction in the fraction of corrupted gradients of more than 90% compared with DETOX [16]. A weaker variation of this attack is where the adversaries do not collude and act randomly. In this case, we demonstrate that the Aspis protocol allows for detecting all the adversaries. In both scenarios, we provide theoretical guarantees on the fraction of corrupted gradients.
- For weaker attacks (discussed above), our results indicate that Aspis+ detects all adversaries within approximately 5 iterations.
- We present top-1 classification accuracy experiments on the CIFAR-10 [32] data set for various gradient distortion attacks coupled with choice/behavior patterns of the adversarial nodes. Under the most sophisticated distortion methods [23], the performance gap between Aspis/Aspis+ and other state-of-the-art methods is substantial, e.g., for Aspis it is 43% in the strong scenario (*cf.* Figure 4a), and for Aspis+, 19% in the weak scenario (*cf.* Figure 10).

III. Distributed Training Formulation

The formulation we discuss is standard in distributed deep learning. Assume a loss function $l_i(\mathbf{w})$ for the i-th sample where $\mathbf{w} \in \mathbb{R}^d$ is the parameter set of the model.[1] We use *mini-batch Stochastic Gradient Descent* (SGD) to minimize the loss over the entire data set, i.e.,

$$\min_{\mathbf{w}} L(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} l_i(\mathbf{w})$$

where n is the data set's size. Initially, $\mathbf{w}$ is randomly set to $\mathbf{w}_0$ ($\mathbf{w}_t$ is the model state at the end of iteration t). A random *batch* B_t of b samples is chosen to perform the update in the t-th iteration. Thus,

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \frac{1}{|B_t|} \sum_{i \in B_t} \nabla l_i(\mathbf{w}_t) \tag{1}$$

where η_t is the learning rate of the t-th iteration. The workers, denoted $U_1, U_2, \ldots, U_K$, compute gradients on subsets of the batch. The training is *synchronous*, i.e., the PS waits for all workers to return before performing an update. It stores the data set and the model and coordinates the protocol.

Task assignment: Each batch B_t is split into f disjoint files $\{B_{t,i}\}_{i=0}^{f-1}$, which are then assigned to the workers according to our placement policy. Computational redundancy is introduced by assigning a given file to $r > 1$ workers. As the load on all the workers is equal, it follows that each worker is responsible for $l = fr/K$ files (l is the *computation load*). We let $\mathcal{N}^w(U_j)$ be the set of files assigned to worker U_j and $\mathcal{N}^f(B_{t,i})$ be the group of workers assigned to file $B_{t,i}$; our placement scheme is such that $\mathcal{N}^f(B_{t,i})$ uniquely identifies the file $B_{t,i}$; thus, we will sometimes refer to the file $B_{t,i}$ by its worker assignment, $\mathcal{N}^f(B_{t,i})$. We will also occasionally use the term *group* (of the assigned workers) to refer to a file. We discuss the actual placement algorithms used in this work in the upcoming subsection III-A.

Training: We will refer to Figure 1 for this exposition. There are $K = 6$ machines and $f = 4$ distinct files (represented by colored circles) replicated $r = 3$ times.[2] Each worker is assigned to $l = 2$ files and computes the sum of gradients (or a distorted value) on each of them. The "**d**" ellipses refer to PS's detection operations immediately after receiving all the gradients.

The algorithm starts with the assignment of files to workers. Subsequently, each worker U_i will compute all l file gradients that involve its assigned files $\mathcal{N}^w(U_i)$ and return them to the PS. In every iteration, the PS will initially run our detection algorithm in an effort to identify the q adversaries and will act according to the detection outcome.

Metrics: We are interested in two main metrics, the fraction of distorted files and the top-1 test accuracy of the final trained model. For the distortion fraction, let us denote the number of distorted tasks upon detection and aggregation by $c^{(q)}$ and its maximum value (under a worst-case attack) by $c^{(q)}_{\max}$. The *distortion fraction* is $\epsilon := c^{(q)}/f$. We evaluate these metrics for the various competing methods.

A. Task Assignment

Let $\mathcal{U}$ be the set of workers. Our scheme has $|\mathcal{U}| \leq f$ (i.e., fewer workers than files). Our assignment of files to worker nodes is specified by a bipartite graph $\mathbf{G}_{task}$ where the left vertices correspond to the workers, and the right vertices correspond to the files. An edge in $\mathbf{G}_{task}$ between worker U_i and a file $B_{t,j}$ indicates that the U_i is responsible for processing file $B_{t,j}$.

[1] The paper's heavily-used notation is summarized in Appendix Table II.

[2] Some arrows and ellipses have been omitted from Figure 1; however, all files will be going through detection.

TABLE I: Adversarial models considered in literature.

Scheme	Byzantine choice/orchestration	Gradient distortion
Draco [17]	random	reversed gradient, constant
DETOX [16]	random	ALIE, reversed gradient, constant
ByzShield [14]	optimal	ALIE, reversed gradient, constant
Bulyan [24]	N/A	ℓ_2-norm attack targeted on Bulyan
Multi-Krum [12]	N/A	random high-variance Gaussian vector
Aspis	ATT-1, ATT-2	ALIE, FoE, reversed gradient
Aspis+	ATT-3	ALIE, constant

1) Aspis: For the Aspis scheme, we construct $\mathbf{G}_{task}$ as follows. The left vertex set is $\{1, 2, \ldots, K\}$ and the right vertex set corresponds to r-sized subsets of $\{1, 2, \ldots, K\}$ (there are $\binom{K}{r}$ of them). An edge between $1 \leq i \leq K$ and $S \subset \{1, 2, \ldots, K\}$ (where $|S| = r$) exists if $i \in S$. The worker set $\{U_1, \ldots, U_K\}$ is in one-to-one correspondence with $\{1, 2, \ldots, K\}$ and the files $B_{t,0}, \ldots, B_{t,f-1}$ are in one-to-one correspondence with the r-sized subsets.

Example 1. Consider $K = 7$ workers $U_1, U_2 \ldots, U_7$ and $r = 3$. Based on our protocol, the $f = \binom{7}{3} = 35$ files of each batch B_t are associated one-to-one with 3-subsets of $\mathcal{U}$, e.g., the subset $S = \{U_1, U_2, U_3\}$ corresponds to file $B_{t,0}$ and will be processed by U_1, U_2, and U_3.

Remark 1. Our task assignment ensures that every pair of workers processes $\binom{K-2}{r-2}$ files. Moreover, the number of adversaries is $q < K/2$. Thus, upon receiving the gradients from the workers, the PS can examine them for consistency and flag certain nodes as adversarial if their computed gradients differ from $q + 1$ or more of the other nodes. We use this intuition to detect and mitigate the adversarial effects and compute the fraction of corrupted files.

2) Aspis+: For Aspis+, we use combinatorial designs [33] to assign the gradient tasks to workers. Formally, a *design* is a pair $(X, \mathcal{A})$ consisting of a set of v elements (*points*), X, and a family $\mathcal{A}$ (i.e., multiset) of nonempty subsets of X called *blocks*, where each block has the same cardinality k. Similar to Aspis, the workers and files are in one-to-one correspondence with the points and the blocks, respectively. Hence, for our purposes, the k parameter of the design is the redundancy. A $t - (v, k, \lambda)$ design is one where any subset of t points appear together in exactly λ blocks. The case of $t = 2$ has been studied extensively in the literature and is referred to as a *balanced incomplete block design* (BIBD). A bipartite graph representing the incidence between the points and the blocks can be obtained naturally by letting the points correspond to the left vertices, and the blocks correspond to the right vertices. An edge exists between a point and a block if the point is contained in the block.

Example 2. A $2 - (7, 3, 1)$ design, also known as the *Fano plane*, consists of the $v = 7$ points $X = \{1, 2, \ldots, 7\}$ and the block multiset $\mathcal{A}$ contains the blocks $\{1, 2, 3\}$, $\{1, 4, 7\}$, $\{2, 4, 6\}$, $\{3, 4, 5\}$, $\{2, 5, 7\}$, $\{1, 5, 6\}$ and $\{3, 6, 7\}$ with each block being of size $k = 3$. In the bipartite graph $\mathbf{G}_{task}$ representation, we would have an edge, e.g., between point 2 and blocks $\{1, 2, 3\}, \{2, 4, 6\}$, and $\{2, 5, 7\}$.

In Aspis+, we construct $\mathbf{G}_{task}$ by the bipartite graph representing an appropriate $2 - (v, k, \lambda)$ design.

Another change compared to the placement of Section III-A is that the points of the design will be randomly permuted at each iteration, i.e., for permutation π, the PS will map $\{U_1, U_2, \ldots, U_K\} \xrightarrow{\pi} \{\pi(U_1), \pi(U_2), \ldots, \pi(U_K)\}$. For instance, let us circularly permute the points of the Fano plane in Example 2 as $\pi(U_i) = U_{i+1}, i = 1, 2, \ldots, K-1$ and $\pi(U_K) = U_1$. Then, the file assignment at the next iteration will be based on the block collection $\mathcal{A} = \{\{2, 3, 4\}, \{1, 2, 5\}, \{3, 5, 7\}, \{4, 5, 6\}, \{1, 3, 6\}, \{2, 6, 7\}, \{1, 4, 7\}\}$. Permuting the assignment causes each Byzantine to disagree with more workers and to be detected in fewer iterations; details will be discussed in Section V-C. Owing to this permutation, we use a time subscript for the files assigned to U_i for the t-th iteration; this is denoted by $\mathcal{N}_t^w(U_i)$.

IV. Adversarial Models

We now discuss the different Byzantine models that we consider in this work. For all the models, we assume that at most $q < K/2$ workers can be adversarial. For each assigned file $B_{t,i}$ a worker U_j will return the value $\hat{\mathbf{g}}_{t,i}^{(j)}$ to the PS. Then,

$$\hat{\mathbf{g}}_{t,i}^{(j)} = \begin{cases} \mathbf{g}_{t,i} & \text{if } U_j \text{ is honest,} \\ * & \text{otherwise,} \end{cases} \tag{2}$$

where $\mathbf{g}_{t,i}$ is the sum of the loss gradients on all samples in file $B_{t,i}$, i.e.,

$$\mathbf{g}_{t,i} = \sum_{j \in B_{t,i}} \nabla l_j(\mathbf{w}_t)$$

and $*$ is any arbitrary vector in $\mathbb{R}^d$. Within this setup, we examine multiple adversarial scenarios in regards to the actual behavior of the workers. Table I provides a high-level summary of the Byzantine models considered in this work, as well as in related papers. As we will discuss in Section VII-B, for those schemes that do not involve redundancy and merely split the work equally among the K workers, all possible choices of the Byzantine set are equivalent, and no *orchestration*[3] of them will change the defense's output; hence, those cases are denoted by "N/A" in the table.

A. Attack 1

We first consider a weak attack, denoted ATT-1, where the Byzantine nodes operate independently (i.e., do not collude) and attempt to distort the gradient on any file they participate in. For instance, a node may try to return arbitrary gradients on all its assigned files. For this attack, the identity of the workers may be arbitrary at each iteration as long as there are at most q of them.

[3]We will use the term *orchestration* to refer to the method adversaries use to collude and attack collectively as a group.

Remark 2. We emphasize that even though we call this attack "weak" this is the attack model considered in several prior works [16], [17]. To our best knowledge, most of them have not considered the adversarial problem from the lens of detection.

B. Attack 2

Our second scenario, named ATT-2, is the strongest one we consider. We assume that the adversaries have full knowledge of the task assignment at each iteration and the detection strategies employed by the PS. The adversaries can collude in the "best" possible way to corrupt as many gradients as possible. Moreover, the set of adversaries can also change from iteration to iteration as long as there are at most q of them.

C. Attack 3

This attack is similar to ATT-1 and will be called ATT-3. On the one hand, it is weaker in the sense that the set of Byzantines A does not change in every iteration. Instead, we will assume that there is a "Byzantine window" of T_b iterations in which the set A remains fixed. Also, the set A will be a randomly chosen set of q workers from $\mathcal{U}$, i.e., it will not be chosen systematically. A new set will be chosen at random at all iterations indexed with t, where $t \equiv 0 \pmod{T_b}$. Conversely, it is stronger than ATT-1 since we allow for limited collusion amongst the adversarial nodes. In particular, the Byzantines simulated by ATT-3 will distort only the files for which a Byzantine majority exists.

V. Defense Strategies in Aspis and Aspis+

We now discuss our proposed Byzantine-resilient distributed training method. It is a best-effort technique for detecting adversaries coupled with robust aggregation.

We propose to use the Aspis task assignment and detection strategy for attacks ATT-1 and ATT-2. For ATT-3, we will use Aspis+. Recall that the methods differ in their corresponding task assignments. Nevertheless, the central idea in both detection methods is for the PS to apply a set of consistency checks on the obtained gradients from the different workers at each iteration to identify the adversaries.

Let the current set of adversaries be $A \subset \{U_1, U_2, \ldots, U_K\}$ with $|A| = q$; also, let H be the honest worker set. The set A is unknown, but our goal is to provide an estimate $\hat{A}$ of it. Ideally, the two sets should be identical. In general, depending on the adversarial behavior, we will be able to provide a set $\hat{A}$ such that $\hat{A} \subseteq A$. For each file, there is a group of r workers which have processed it, and there are $\binom{r}{2}$ pairs of workers in each group. Each such pair may or may not agree on the gradient value for the file. For an iteration, let us encode the agreement of workers U_{j_1}, U_{j_2} on common file i during the current iteration t by

$$\alpha_{t,i}^{(j_1,j_2)} := \begin{cases} 1 & \text{if } \hat{\mathbf{g}}_{t,i}^{(j_1)} = \hat{\mathbf{g}}_{t,i}^{(j_2)}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Across all files, the total number of agreements between a pair of workers U_{j_1}, U_{j_2} during the t-th iteration is denoted by

$$\alpha_t^{(j_1,j_2)} := \sum_{i \in \mathcal{N}_t^w(U_{j_1}) \cap \mathcal{N}_t^w(U_{j_2})} \alpha_{t,i}^{(j_1,j_2)}. \tag{4}$$

Algorithm 1: Proposed Aspis graph-based detection.

Input: Computed gradients $\hat{\mathbf{g}}_{t,i}^{(j)}$, $i = 0, 1, \ldots, f-1$, $j = 1, 2, \ldots, K$, redundancy r and empty graph $\mathbf{G}$ with worker vertices $\mathcal{U}$.

for *each pair* $(U_{j_1}, U_{j_2}), j_1 \neq j_2$ *of workers* **do**
  PS computes the number of agreements $\alpha^{(j_1,j_2)}$ of the pair U_{j_1}, U_{j_2} on the gradient value.
  if $\alpha^{(j_1,j_2)} = \binom{K-2}{r-2}$ **then**
    Connect vertex U_{j_1} to vertex U_{j_2} in $\mathbf{G}$.
  end
end
PS enumerates all k maximum cliques $M_{\mathbf{G}}^{(1)}, M_{\mathbf{G}}^{(2)}, \ldots, M_{\mathbf{G}}^{(k)}$ in $\mathbf{G}$.
if *there is a unique maximum clique* $M_{\mathbf{G}}$ *(*$k = 1$*)* **then**
  PS determines the honest workers $H = M_{\mathbf{G}}$ and the adversarial machines $\hat{A} = \mathcal{U} - M_{\mathbf{G}}$.
else
  PS declares unsuccessful detection.
end

Since the placement is known, the PS can always perform the above computation. Next, we form an undirected graph $\mathbf{G}_t$ whose vertices correspond to all workers $\{U_1, U_2, \ldots, U_K\}$. An edge (U_{j_1}, U_{j_2}) exists in $\mathbf{G}_t$ only if the computed gradients (at iteration t) of U_{j_1} and U_{j_2} match in "all" their common assignments.

A. Aspis Detection Rule

In what follows, we suppress the iteration index t since the Aspis algorithm is the same for each iteration. For the Aspis task assignment (*cf.* Section III-A1), any two workers, U_{j_1} and U_{j_2}, have $\binom{K-2}{r-2}$ common files.

Let us index the q adversaries in $A = \{A_1, A_2, \ldots, A_q\}$ and the honest workers in H. We say that two workers U_{j_1} and U_{j_2} disagree if there is no edge between them in $\mathbf{G}$. The non-existence of an edge between U_{j_1} and U_{j_2} only means that they disagree in *at least one* of the $\binom{K-2}{r-2}$ files that they jointly participate in. For corrupting the gradients, each adversary has to disagree on the computations with a subset of the honest workers. An adversary may also disagree with other adversaries. Let D_i denote the set of disagreement workers for adversary $A_i, i = 1, 2, \ldots, q$, where D_i can contain members from A and from H.

A *clique* in an undirected graph is defined as a subset of vertices with an edge between any pair of them. A *maximal clique* is one that cannot be enlarged by adding additional vertices to it. A *maximum clique* is one such that there is no clique with more vertices in the given graph. We note that the set of honest workers H will pair-wise agree everywhere. In particular, this implies that the subset H forms a clique (of size $K - q$) within $\mathbf{G}$. The clique containing the honest workers may not be maximal. However, it will have a size of at least $K - q$. Let the maximum clique on $\mathbf{G}$ be $M_{\mathbf{G}}$. Any worker U_j with $\deg(U_j) < K - q - 1$ will not belong to a maximum clique and can right away be eliminated as a "detected" adversary.

The essential idea of our detection is to run a clique-finding algorithm on $\mathbf{G}$ (summarized in Algorithm 1). The detection

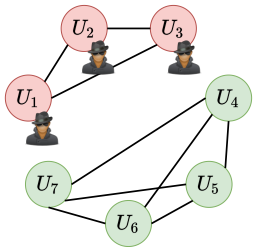
Algorithm 2: Proposed Aspis/Aspis+ aggregation protocol to alleviate Byzantine effects.

Input: Data set of n samples, batch size b, computation load l, redundancy r, number of files f, maximum iterations T, file assignments $\{\mathcal{N}^w(U_i)\}_{i=1}^K$.

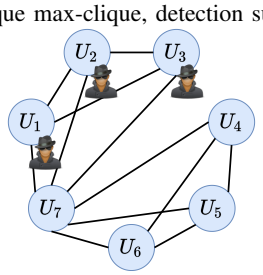
The PS randomly initializes the model's parameters to $\mathbf{w}_0$.
for $t = 0$ *to* $T-1$ **do**
  PS chooses a random batch $B_t \subseteq \{1, 2, \ldots, n\}$ of b samples, partitions it into f files $\{B_{t,i}\}_{i=0}^{f-1}$ and assigns them to workers according to $\{\mathcal{N}^w(U_i)\}_{i=1}^K$. It then transmits $\mathbf{w}_t$ to all workers.
  for *each worker* U_j **do**
    if U_j *is honest* **then**
      for *each file* $i \in \mathcal{N}^w(U_j)$ **do**
        U_j computes the sum of gradients
        $$\hat{\mathbf{g}}_{t,i}^{(j)} = \sum_{k \in B_{t,i}} \nabla l_k(\mathbf{w}_t).$$
      end
    else
      U_j constructs l adversarial vectors
      $$\hat{\mathbf{g}}_{t,i_1}^{(j)}, \hat{\mathbf{g}}_{t,i_2}^{(j)}, \ldots, \hat{\mathbf{g}}_{t,i_l}^{(j)}.$$
    end
    U_j returns $\hat{\mathbf{g}}_{t,i_1}^{(j)}, \hat{\mathbf{g}}_{t,i_2}^{(j)}, \ldots, \hat{\mathbf{g}}_{t,i_l}^{(j)}$ to the PS.
  end
  PS runs a detection algorithm to identify the adversaries.
  if *detection is successful* **then**
    Let H be the detected honest workers. Initialize a non-corrupted gradient set as $\mathcal{G} = \emptyset$.
    for *each file in* $\{B_{t,i}\}_{i=0}^{f-1}$ **do**
      PS chooses the gradient of a worker in $\mathcal{N}^f(B_{t,i}) \cap H$ (if non-empty) and adds it to $\mathcal{G}$.
    end
    $$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \frac{1}{|\mathcal{G}|} \sum_{\mathbf{g} \in \mathcal{G}} \mathbf{g}.$$
  else
    for *each file in* $\{B_{t,i}\}_{i=0}^{f-1}$ **do**
      PS determines the r workers in $\mathcal{N}^f(B_{t,i})$ which have processed $B_{t,i}$ and computes $\mathbf{m}_i = \text{majority}\left\{\hat{\mathbf{g}}_{t,i}^{(j)} : U_j \in \mathcal{N}^f(B_{t,i})\right\}$.
    end
    PS updates the model via $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \times \text{median}\{\mathbf{m}_i : i = 0, 1, \ldots, f-1\}$.
  end
end

may be successful or unsuccessful depending on whether ATT-1 or ATT-2 has been used. The PS will determine the situation and act differently depending on the case. We will show that if ATT-1 is used, detection will be successful, while if ATT-2 is used, detection will be unsuccessful. The two different cases are discussed in the following subsections V-A1 and V-A2.

We note that clique-finding is well-known to be an NP-complete problem [34]. Nevertheless, there are fast, practical algorithms with excellent performance on graphs even up to hundreds of nodes [35], [36]. Specifically, the authors of [36] have shown that their proposed algorithm, which enumerates all maximal cliques, has similar complexity as other methods [37], [38], which are used to find a single maximum clique. We utilize this algorithm. Our extensive experimental evidence suggests that clique-finding is not a computation bottleneck for the size and structure of the graphs that Aspis uses. We have experimented with clique-finding on a graph of $K = 100$ workers and $r = 5$ for different values of q; in all cases, enumerating all maximal cliques took no more than 15 milliseconds. These experiments and the asymptotic complexity of the entire protocol are addressed in Supplement Section X-A.



(a) Unique max-clique, detection succeeds.



(b) Two max-cliques, detection fails.

Fig. 2: Detection graph $\mathbf{G}$ for $K = 7$ workers among which U_1, U_2 and U_3 are the adversaries.

The details of the entire aggregation procedure are described in Algorithm 2. As our methods rely on gradient equality checks, we ensure that no floating-point precision issues occur in our implementation if run on CPUs, i.e., all honest workers assigned to $B_{t,i}$ will return the exact same gradient. On GPUs, as NVIDIA CUDA [39] is non-deterministic, negligible precision issues in the order of at most 10^{-6} may occur. Even in this case, we can easily handle the issue using a tolerance-based equality check and the ℓ^2 norm (*cf.* Supplement Section X-B).

1) Defense Strategy Against ATT-1: This is the case where Aspis succeeds in detecting the Byzantines. Under ATT-1, it is clear that the Byzantine node will disagree with at least $K - q$ honest nodes (as, by assumption in Section IV-A, it will disagree with all of them), and thus, the degree of the node in $\mathbf{G}$ will be at most $q - 1 < K - q - 1$, and it will not be part of the maximum clique. Thus, each of the adversaries will be detected, and their returned gradients will not be considered further. The algorithm declares the (unique) maximum clique as honest and proceeds to aggregation. In particular, assume that h workers $U_{i_1}, U_{i_2}, \ldots, U_{i_h}$ have been identified as honest. For each of the f files, if at least one honest worker processed it, the PS will pick one of the "honest" gradient values. The chosen gradients are then

averaged for the update (*cf.* Eq. (1)). For instance, in Figure 1, assume that U_1, U_2, and U_4 have been identified as faulty. During aggregation, the PS will ignore the red file as all 3 copies have been compromised. For the orange file, it will pick either the gradient computed by U_5 or U_6 as both of them are "honest." The only files that can be distorted in this case are those that consist exclusively of adversarial nodes.

Figure 2a (corresponding to Example 1) shows an example where in a cluster of size $K = 7$, the $q = 3$ adversaries are $A = \{U_1, U_2, U_3\}$ and the remaining workers are honest with $H = \{U_4, U_5, U_6, U_7\}$. In this case, the unique maximum clique is $M_{\mathbf{G}} = H$, and detection is successful. Under this attack, the distorted files are those whose all copies have been compromised, i.e., $c^{(q)} = \binom{q}{r}$.

2) Defense Strategy Against ATT-2: If the attack ATT-2 is used on Aspis, upon the formation of $\mathbf{G}$, we know that a worker U_j will be flagged as adversarial if $deg(U_j) < K - q - 1$. Therefore to avoid detection, a *necessary* condition is that $|D_j| \leq q$. We now upper bound the number of files that can be corrupted under any possible strategy employed by the adversaries. Note that according to Algorithm 2, we resort to robust aggregation in case of more than one maximum clique in $\mathbf{G}$. In this scenario, a gradient can only be corrupted if a majority of the assigned workers computing it are adversarial and agree on a wrong value.

For a given file F, let $A' \subseteq A$ with $|A'| \geq r'$ be the set of "active adversaries" in it, i.e., $A' \subseteq F$ consists of Byzantines that collude to create a majority that distorts the gradient on it. In this case, the remaining workers in F belong to $\cap_{i \in A'} D_i$, where we note that $|\cap_{i \in A'} D_i| \leq q$. Let $X_j, j = r', r'+1 \ldots, r$ denote the subset of files where the set of active adversaries is of size j; note that X_j depends on the disagreement sets $D_i, i = 1, 2, \ldots, q$. Formally,

$$X_j = \{F : \exists A' \subseteq A \cap F, |A'| = j, \text{ and } \forall\ U_j \in F \setminus A', U_j \in \cap_{i \in A'} D_i\}. \quad (5)$$

Then, for a given choice of disagreement sets, the number of files that can be corrupted is given by $|\cup_{j=r'}^{r} X_j|$. We obtain an upper bound on the maximum number of corrupted files by maximizing this quantity with respect to the choice of $D_i, i = 1, 2, \ldots, q$, i.e.,

$$c_{\max}^{(q)} = \max_{D_i, |D_i| \leq q, i = 1,2,\ldots,q} |\cup_{j=r'}^{r} X_j| \quad (6)$$

where the maximization is over the choices of the disagreement sets $D_1, D_2, \ldots, D_q$. The proof of the following theorem appears in Appendix Section IX-A.

Theorem 1. Consider a training cluster of K workers with q adversaries using the algorithm in Section III-A1 to assign the $f = \binom{K}{r}$ files to workers, and Algorithm 1 for adversary detection. Under an optimal adversary model, the maximum number of files that can be corrupted is

$$c_{\max}^{(q)} = \frac{1}{2}\binom{2q}{r}. \quad (7)$$

Furthermore, this upper bound can be achieved if all adversaries fix a set $D \subset H$ of honest workers with which they will consistently disagree on the gradient (by distorting it).

In particular, the proposed attack is optimal, and there is no other attack that can corrupt more files under the Aspis algorithm. One such attack is carried out in Figure 2b for the setup of Example 1. The adversaries $A = \{U_1, U_2, U_3\}$ consistently disagree with the workers in $D = \{U_4, U_5, U_6\} \subset H$. The ambiguity as to which of the two maximum cliques ($\{U_1, U_2, U_3, U_7\}$ or $\{U_4, U_5, U_6, U_7\}$) is the honest one makes an accurate detection impossible; robust aggregation will be performed instead.

In Appendix Section IX-A, we show that ATT-2 is guaranteed to form more than one maximum clique on the detection graph. Thus, the PS cannot unambiguously decide which one is the honest one; detection fails, and we fall back to the robust aggregation technique. During aggregation, the PS will perform a majority vote across the computations of each file. Recall that r workers have processed each file. For each such file $B_{t,i}$, the PS decides a majority value $\mathbf{m}_i$

$$\mathbf{m}_i := \text{majority}\left\{\hat{\mathbf{g}}_{t,i}^{(j)} : U_j \in \mathcal{N}^f(B_{t,i})\right\}. \quad (8)$$

Assume that r is odd and let $r' = \frac{r+1}{2}$. Under the rule in Eq. (8), the gradient on a file is distorted only if at least r' of the computations are performed by Byzantines. Following the majority vote, we will further filter the gradients using coordinate-wise median and refer to the combination of these two steps as *robust aggregation*; a similar setup was considered in [14], [16]. For example, in Figure 1, all returned values for the red file will be evaluated by a majority vote function on the PS, which decides a single output value; similar voting is done for the other 3 files. After the voting process, Aspis applies coordinate-wise median on the "winning" gradients $\mathbf{m}_i$, $i = 0, 1, \ldots, f-1$.

B. Motivation for Aspis+

Our motivation for proposing Aspis+ originates in the limitations of the subset assignment of Aspis. It is evident from the experimental results in following Section VII-B that Aspis is more suitable to worst-case attacks where the adversaries collude and distort the maximum number of tasks undetected; in this case, the accuracy gap between Aspis and prior methods is maximal. Aspis does not perform as well under weaker attacks such as the *reversed gradient* attack (*cf.* Figures 5a, 5b, 5c even though it achieves a much smaller distortion fraction ϵ, as discussed in Section VI. This can be attributed to the fact that the number of tasks is $\binom{K}{r}$ and even for the considered cluster of $K = 15$, $r = 3$, it would require splitting the batch into 455 files; hence, the batch size must be a multiple of 455. There is significant evidence that large batch sizes can hurt generalization and make the model converge slowly [31], [40], [41]. Some workarounds have been proposed to solve this problem. For instance, the work of [41] uses layer-wise adaptive rate scaling to update each layer using a different learning rate. The authors of [42] perform implicit regularization using warmup and cosine annealing to tune the learning rate as well as gradient clipping. However, these methods require training for a significantly larger number of epochs. For the above reasons, we have extended our work and proposed Aspis+ to handle weaker Byzantine failures (*cf.* ATT-3) while requiring a much smaller batch size.

Algorithm 3: Proposed Aspis+ graph-based detection.

Input: Computed gradients $\hat{\mathbf{g}}_{t,i}^{(j)}$, $i = 0, 1, \ldots, f-1$, $j = 1, 2, \ldots, K$, $2-(v, k, \lambda)$ design, length of detection window T_d, maximum iterations T.

for $t = 0$ *to* $T-1$ **do**
  Let $t' = t \pmod{T_d} + 1$.
  if $t' = 1$ **then**
    Set $\mathbf{G}$ as the complete graph with worker vertices $\mathcal{U}$.
    $\forall j_1, j_2$, set $\alpha^{(j_1,j_2)} = 0$.
  end
  for *each pair* $(U_{j_1}, U_{j_2}), j_1 \neq j_2$ *of workers* **do**
    PS computes the number of agreements $\alpha_t^{(j_1,j_2)}$ of the pair U_{j_1}, U_{j_2} on the gradient value.
    Update $\alpha^{(j_1,j_2)} = \alpha^{(j_1,j_2)} + \alpha_t^{(j_1,j_2)}$.
  end
  for *each pair* $(U_{j_1}, U_{j_2}), j_1 \neq j_2$ *of workers* **do**
    if $\alpha^{(j_1,j_2)} < \lambda \times t'$ **then**
      Remove edge (U_{j_1}, U_{j_2}) from $\mathbf{G}$.
    end
  end
  for *each worker* $U_j \in \mathcal{U}$ **do**
    if $deg(U_j) < K - q - 1$ **then**
      $\hat{A} = \hat{A} \cup \{U_j\}$.
    end
  end
  if $|\hat{A}| > q$ **then**
    Set $\hat{A}$ to be the q most recently detected Byzantines.
  end
end

C. Aspis+ Detection Rule

Note that ATT-3 is the weakest attack model that we are considering. The principal difference with ATT-1 is that the set of Byzantine workers is only allowed to change every few iterations. Moreover, the adversaries are chosen randomly rather than adversarially. However, we emphasize that this may be the most realistic model, and several prior works only consider this attack model.

The principal intuition of the Aspis+ detection approach (used for ATT-3) is to iteratively keep refining the graph $\mathbf{G}_t$ in which the edges encode the agreements of workers during consecutive and non-overlapping windows of T_d iterations. At the beginning of each such window, the PS will reset $\mathbf{G}$ to be a complete graph, i.e., as if all workers pairwise agree with other. Then, it will gradually remove edges from $\mathbf{G}$ as disagreements between the workers are observed; hence, the graph will be updated at each of the T_d iterations of the window, and the PS will assume that the Byzantine set does not change within a detection window. In practice, as we do not know the "Byzantine window," we will not assume an alignment between the two kinds of windows, and we will set $T_d \neq T_b$ for our experiments. The detection method will be the same for all detection windows; thus, we will analyze the process in one window of T_d steps.

For a detection window, let us encode the agreement of workers U_{j_1}, U_{j_2} on a common file i during the current iteration t of the window $t = 1, 2, \ldots, T_d$ as

$$\alpha_{t,i}^{(j_1,j_2)} := \begin{cases} 1 & \text{if } \hat{\mathbf{g}}_{t,i}^{(j_1)} = \hat{\mathbf{g}}_{t,i}^{(j_2)}, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Across all files, the total number of agreements between a pair of workers U_{j_1}, U_{j_2} during the t-th iteration is denoted by

$$\alpha_t^{(j_1,j_2)} := \sum_{i \in \mathcal{N}_t^w(U_{j_1}) \cap \mathcal{N}_t^w(U_{j_2})} \alpha_{t,i}^{(j_1,j_2)}. \tag{10}$$

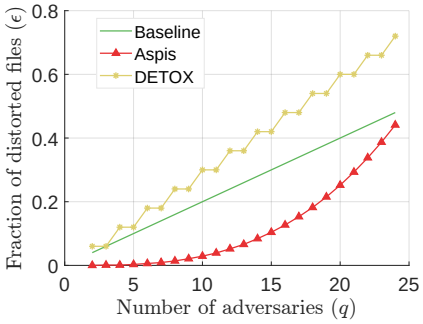
Assume that the current iteration of the window is indexed with $t' \in \{1, 2, \ldots, T_d\}$. The PS will collect all agreements for each pair of workers U_{j_1}, U_{j_2} up until the current iteration as

$$\alpha^{(j_1,j_2)} := \sum_{t=1}^{t'} \alpha_t^{(j_1,j_2)}. \tag{11}$$

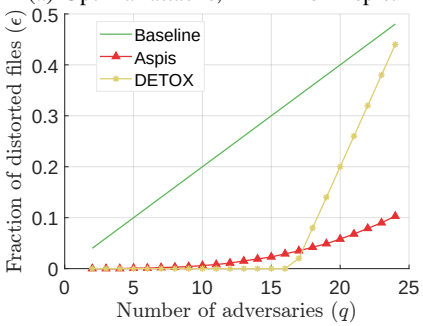
Since the placement is known, the PS can always perform the above computation. Next, it will examine the agreements and update $\mathbf{G}$ as necessary.

Based on the task placement (*cf.* Section III-A2), an edge (U_{j_1}, U_{j_2}) exists in $\mathbf{G}$ only if the computed gradients of U_{j_1} and U_{j_2} match in all their λ common groups in all iterations up to the current one indexed with t', i.e., a pair U_{j_1}, U_{j_2} needs to have $\alpha^{(j_1,j_2)} = \lambda \times t'$ for an edge (U_{j_1}, U_{j_2}) to be in $\mathbf{G}$. If this is not the case, the edge (U_{j_1}, U_{j_2}) will be removed from $\mathbf{G}$. After all such edges are examined, detection is done using degree counting. Given that there are q Byzantines in the cluster, after examining all pairs of workers and determining the form of $\mathbf{G}$, a worker U_j will be flagged as Byzantine if $deg(U_j) < K - q - 1$. Based on Eq. (11), it is not hard to see that such workers can be eliminated, and their gradients will not be considered again until the last iteration of the current window. The only exception to this is if the Byzantine set changes before the end of the current detection window. This is possible due to a potential misalignment between the "Byzantine window" and the detection window (recall that $T_d \neq T_b$ is assumed to avoid trivialities). In this case, more than q workers may be detected as Byzantines; the PS will, by convention, choose $\hat{A}$ to be the most recently detected Byzantines. Algorithm 3 discusses the detection protocol. Following detection, the PS will act as follows. If at least one Byzantine has been detected, it will ignore the votes of detected Byzantines, and for each group, if there is at least one "honest" vote, it will use this as the output of the majority voting group; also, if a group consists merely of detected Byzantines, it will completely ignore the group. The remaining groups will go through robust aggregation (as in Section V-A). In our experiments in Section VII-C, all Byzantines are detected successfully in at most 5 iterations. Example 3 showcases the utility of permutations in our detection algorithm using $K = 7$ workers.

Example 3. We will use the assignment of Example 2 with $K = 7$ workers $\mathcal{U} = \{1, 2, \ldots, 7\}$ assigned to tasks according to a $2-(7, 3, 1)$ Fano plane, and let us denote the assignment of workers to groups (blocks of the design) during the t-th iteration by $\mathcal{A}_t$, initially equal to $\mathcal{A}_1 = \{\{1, 2, 3\}, \{1, 4, 7\}, \{2, 4, 6\}, \{3, 4, 5\}, \{2, 5, 7\}, \{1, 5, 6\}, \{3, 6, 7\}\}$. For the windows, assume that $T_d > 2$ and $T_b > 2$. Also, let $q = 2$ and the Byzantine set be $A = \{U_1, U_2\}$. Based on ATT-3, workers U_1, U_2 are in majority within a group in which



(a) Optimal attacks, ATT-2 for Aspis.



(b) Weak attacks, ATT-1 for Aspis.

Fig. 3: Distortion fraction of optimal and weak attacks for $(K, r) = (50, 3)$ and comparison.

they disagree with worker U_3. After the first permutation, a possible assignment is $\mathcal{A}_2 = \{\{1,3,6\},\{3,4,7\},\{2,4,6\},\{1,4,5\},\{5,6,7\},\{2,3,5\},\{1,2,7\}\}$. Then, U_1 and U_2 are in the same group as the honest U_7 with which they disagree; hence, $deg(U_1) = deg(U_2) = 4 = K - q - 1$, and none of them affords to disagree with more honest workers to remain undetected. However, if the next permutation assigns the workers as $\mathcal{A}_3 = \{\{1,3,6\},\{1,4,7\},\{4,6,5\},\{2,3,4\},\{2,6,7\},\{1,2,5\},\{3,5,7\}\}$ then the adversaries will cast a different vote than U_5 as well. Both of them will be detected after only three iterations.

Remark 3. Using a $2-(v,3,\lambda)$, i.e., a design with $k = 3$ (a typical value for the redundancy) to assign the files on a cluster with q Byzantines, the maximum number of files one can distort is $\lambda\binom{q}{2}/|\mathcal{B}|$ [33], where $|\mathcal{B}|$ is the total number of files; this is when each possible pair of Byzantines, among the $\binom{q}{2}$ possible ones appear together in a distinct block and distort the corresponding file. In Aspis+, the focus is on weak attacks, and determining the worst-case choice of adversaries that maximize the number of distorted files is beyond the scope of our work.

VI. Distortion Fraction Evaluation

We have performed simulations of the fraction of distorted files (defined as $\epsilon = c^{(q)}/f$) incurred by Aspis and other competing methods. The main motivation of this analysis is that our deep learning experiments (*cf.* Section VII-B) and prior work [14] show that ϵ is a surrogate of the model's convergence with respect to accuracy. In addition, our simulations show that Aspis enjoys values of ϵ, which are as much as 99% lower for the same q compared to other techniques, and this attests to our theoretical robustness guarantees. This comparison involves our work and state-of-the-art schemes under the best- and worst-case choice of the q adversaries in terms of the achievable value of ϵ. We also compare our work with *baseline* approaches that do not involve redundancy or majority voting. Their aggregation is applied directly to the K gradients returned by the workers ($f = K$, $c^{(q)}_{\max} = q$ and $\epsilon = q/K$).

Let us first discuss the scenario of an *optimal* attack. For Aspis, we used the proposed attack ATT-2 from Section IV-B and the corresponding computation of $c^{(q),Aspis}$ of Theorem 1. *DETOX* in [16] employs a redundant assignment followed by majority voting and offers robustness guarantees which crucially rely on a "random choice" of the Byzantines. Our prior work [14] (ByzShield) has demonstrated the importance of a careful task assignment and observed that redundancy by itself is not sufficient to allow for Byzantine resilience. That work proposed an optimal choice of the q Byzantines that maximizes ϵ^{DETOX}, which we used in our current experiments. In short, DETOX splits the K workers into K/r groups. All workers within a group process the same subset of the batch, specifically containing br/K samples. This phase is followed by majority voting on a group-by-group basis. The authors of [14] suggested choosing the Byzantines so that at least r' workers in each group are adversarial in order to distort the corresponding gradients. In this case, $c^{(q),DETOX} = \lfloor \frac{q}{r'} \rfloor$ and $\epsilon^{DETOX} = \lfloor \frac{q}{r'} \rfloor \times r/K$. We also compare with the distortion fraction incurred by ByzShield [14] under a worst-case scenario. For this scheme, there is no known optimal attack, and we performed an exhaustive combinatorial search to find the q adversaries that maximize $\epsilon^{ByzShield}$ among all possible options; we follow the same process here to simulate ByzShield's distortion fraction computation while utilizing the scheme of that work based on *mutually orthogonal Latin squares*. The reader can refer to Figure 3a and Appendix Tables III, IV, and V for our results. Aspis achieves major reductions in ϵ; for instance, $\epsilon^{Aspis,ATT-2}$ is reduced by up to 99% compared to both $\epsilon^{Baseline}$ and ϵ^{DETOX} in Figure 3a.

Next, we consider the *weak* attack, ATT-1. For our scheme, we will make an arbitrary choice of q adversaries which carry out the method introduced in Section V-A1, i.e., they will distort all files and a successful detection is possible. As discussed in Section V-A1, the fraction of corrupted gradients is $\epsilon^{Aspis,ATT-1} = \binom{q}{r}/\binom{K}{r}$. For DETOX, a simple benign attack is used. To that end, let the K/r files be $B_{t,0}, B_{t,1}, \ldots, B_{t,K/r-1}$. Initialize $A = \emptyset$ and choose the q Byzantines as follows: for $i = 0, 1, \ldots, q-1$, among the remaining workers in $\{U_1, U_2, \ldots, U_K\} - A$ add a worker from the group $B_{t,i \mod K/r}$ to the adversarial set A. Then,

$$c^{(q),DETOX} = \begin{cases} q - \frac{K}{r}(r'-1) & \text{if } q > \frac{K}{r}(r'-1), \\ 0 & \text{otherwise.} \end{cases}$$

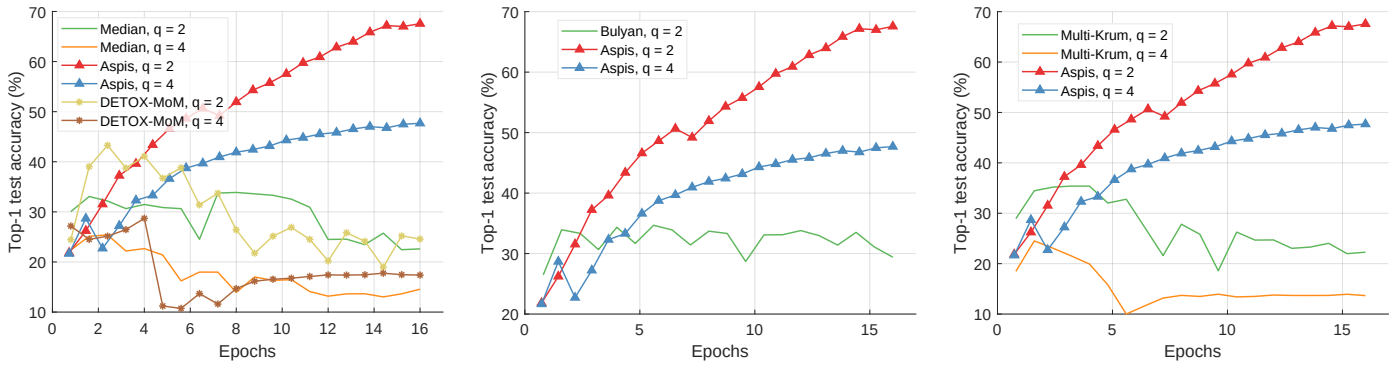
The results of this scenario are in Figure 3b.

For baseline schemes, there is no notion of "weak" or "optimal" attack concerning the choice of the q Byzantines; hence we can choose any subset of them achieving $\epsilon^{Baseline} = q/K$.

VII. Large-Scale Deep Learning Experiments

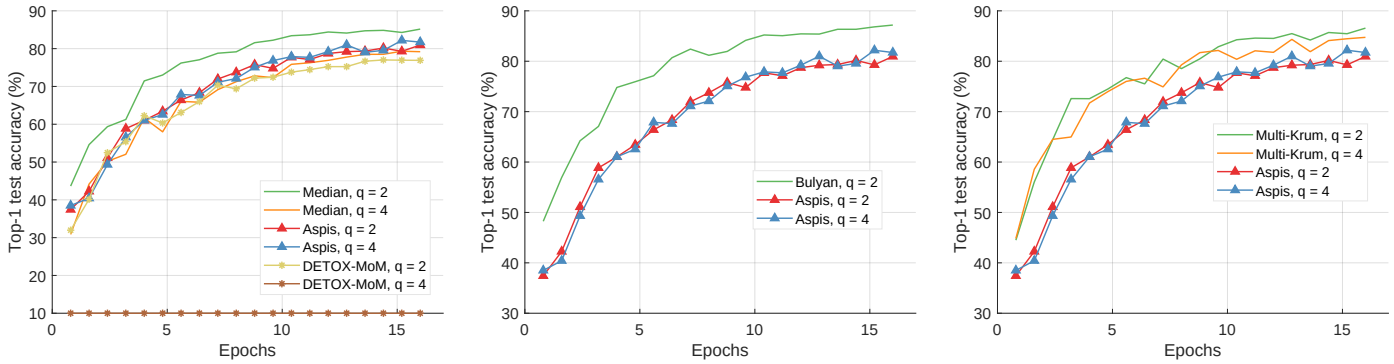
A. Experiment Setup

We have evaluated the performance of our methods and competing techniques in classification tasks on Amazon EC2



(a) Median-based defenses. (b) *Bulyan*-based defenses. (c) *Multi-Krum*-based defenses.

Fig. 4: *ALIE* distortion under optimal attack scenarios, ATT-2 for Aspis, CIFAR-10, $K = 15$.



(a) Median-based defenses. (b) *Bulyan*-based defenses. (c) *Multi-Krum*-based defenses.

Fig. 5: *Reversed gradient* distortion under optimal attack scenarios, ATT-2 for Aspis, CIFAR-10, $K = 15$.

clusters. The project is written in PyTorch [1] and uses the MPICH library for communication between the different nodes. We worked with the CIFAR-10 data set [32] using the ResNet-18 [43] model. We used clusters of $K = 15$, 21, 25 workers, redundancy $r = 3$, and simulated values of $q = 2, 4, 6, 7, 9$ during training. Detailed information about the implementation can be found in Appendix Section IX-B.

There are two different dimensions along which we performed the experimental evaluation.

1) Choice/orchestration of the adversaries: This involves the different ways in which adversaries are chosen and can work together to inflict damage (*cf.* different attacks in Section IV).

2) Gradient distortion methods: This dimension is concerned with the method an adversary uses to distort the gradient value. For instance, *ALIE* [23] involves communication among the Byzantines in which they jointly estimate the mean μ_i and standard deviation σ_i of the batch's gradient for each dimension i and subsequently use them to construct a distorted gradient that attempts to distort the median of the results. Another powerful attack is the *Fall of Empires (FoE)* [44] which performs "inner product manipulation" to make the inner product between the true gradient and the robust estimator to be negative even when their distance is upper bounded by a small value. *Reversed gradient* distortion returns $-c\mathbf{g}$ for $c > 0$ to the PS instead of the true gradient $\mathbf{g}$. The *constant attack* involves the Byzantine workers sending a constant matrix with all elements equal to a fixed value; the matrix has the same dimensions as the true gradient. To our best knowledge, the ALIE algorithm is the most sophisticated attack in literature.

Competing methods: We compare Aspis against the baseline implementations of median-of-means [45], Bulyan [24], and Multi-Krum [12]. If $c_{\max}^{(q)}$ is the number of adversarial computations, then Bulyan requires at least $4c_{\max}^{(q)} + 3$ total number of computations while the same number for Multi-Krum is $2c_{\max}^{(q)} + 3$. These constraints make these methods inapplicable for larger values of q for which our methods are robust. The second class of comparisons is with methods that use redundancy, specifically DETOX [16], for which we show that it can easily fail under malicious scenarios for large q. We compare with median-based techniques since they originate from robust statistics and are the basis for many aggregators. DETOX is the most related redundancy-based work that is based on coding-theoretic techniques. Finally, Multi-Krum is a highly-cited aggregator that combines the intuitions of majority-based and squared-distance-based methods. Draco [17] is a closely related method that uses redundancy; the reason that we chose not to compare with it resides in the very limited fraction of Byzantines it is robust against (*cf.* Section I).

Note that for a baseline scheme, all choices of A are equivalent in terms of the value of ϵ. In our comparisons between Aspis and DETOX, we will consider two attack scenarios concerning the choice of the adversaries. For the *optimal* choice in DETOX, we will use the method proposed in [14] and compare with the attack introduced in Section V-A2. For the *weak* one, we will choose the adversaries such that ϵ is minimized in DETOX and compare its performance with the

scenario of Section V-A1. All schemes compared with Aspis+ consider random sets of Byzantines and for Aspis+, we will use the attack ATT-3.

B. Aspis Experimental Results

1) Comparison under Optimal Attacks: We compare the different defense algorithms under optimal attack scenarios using ATT-2 for Aspis. Figure 4a compares our scheme Aspis with the baseline implementation of coordinate-wise median ($\epsilon = 0.133, 0.267$ for $q = 2, 4$, respectively) and DETOX with median-of-means ($\epsilon = 0.2, 0.4$ for $q = 2, 4$, respectively) under the ALIE attack. Aspis converges faster and achieves at least a 35% average accuracy boost (at the end of the training) for both values of q ($\epsilon^{Aspis} = 0.004, 0.062$ for $q = 2, 4$, respectively).[4] In Figures 4b and 4c, we observe similar trends in our experiments with Bulyan and Multi-Krum, where Aspis significantly outperforms these techniques. For the current setup, Bulyan is not applicable for $q = 4$ since $K = 15 < 4c_{\max}^{(q)} + 3 = 4q + 3 = 19$. Also, neither Bulyan nor Multi-Krum can be paired with DETOX for $q \geq 1$ since the inequalities $f \geq 4c_{\max}^{(q)} + 3$ and $f \geq 2c_{\max}^{(q)} + 3$ cannot be satisfied. Please refer to Section VII-A and Section VI for more details on these requirements. Also, note that the accuracy of most competing methods fluctuates more than in the results presented in the corresponding papers [16] and [23]. This is expected as we consider stronger attacks than those papers, i.e., optimal deterministic attacks on DETOX and, in general, up to 27% of adversarial workers in the cluster. Also, we have done multiple experiments with different random seeds to demonstrate the stability and superiority of our accuracy results compared to other methods (against median-based defenses in Appendix Figure 12, Bulyan in Figure 13 and Multi-Krum in Supplement Figure 14); we point the reader to Appendix Section IX-B3 for this analysis. This analysis is clearly missing from most prior work, including that of ALIE [23], and their presented results are only a snapshot of a single experiment that may or may not be reproducible as is. The results for the reversed gradient attack are shown in Figures 5a, 5b, and 5c. Given that this is a much weaker attack [14], [16], all schemes, including the baseline methods, are expected to perform well; indeed, in most cases, the model converges to approximately 80% accuracy. However, DETOX fails to converge to high accuracy for $q = 4$ as in the case of ALIE; one explanation is that $\epsilon^{DETOX} = 0.4$ for $q = 4$. Under the Fall of Empires (FoE) distortion (*cf.* Figure 6), our method still enjoys an accuracy advantage over the baseline and DETOX schemes which becomes more important as the number of Byzantines in the cluster increases.

We have also performed experiments on larger clusters ($K = 21$ workers) as well. The results for the ALIE distortion with the ATT-2 attack can be found in Figure 9. They exhibit similar behavior as in the case of $K = 15$.

2) Comparison under Weak Attacks: For baseline schemes, the discussion of weak versus optimal choice of the adversaries is not very relevant as any choice of the q Byzantines can overall distort at most q out of the K gradients. Hence, for weak scenarios, we chose to compare mostly with DETOX while using ATT-1 on Aspis. The accuracy is reported in Figures 7 and 8, according to which Aspis shows an improvement under attacks on the more challenging end of the spectrum (ALIE). According to Appendix Table III(b), Aspis enjoys a fraction $\epsilon^{Aspis} = 0.044$ while $\epsilon^{Baseline} = 0.4$ and $\epsilon^{DETOX} = 0.2$ for $q = 6$.

[4]Please refer to Appendix Tables III and IV for the values of the distortion fraction ϵ each scheme incurs.

C. Aspis+ Experimental Results

For Aspis+, we considered the failure scenario ATT-3 discussed in Section IV-C. We tested clusters of $K = 15$ with $q = 2, 4$ and $K = 25$ workers among which $q = 7, 9$ are Byzantine. In the former case, a $2-(15, 3, 1)$ design [33] with $f = 35$ blocks (files) was used for the placement, while in the latter case, we used a $2-(25, 3, 1)$ design [33] with $f = 100$ blocks (files). A new random Byzantine set A is generated every $T_b = 50$ iterations while the detection window is of length $T_d = 15$.

The results for $K = 15$ are in Figure 10. We tested against the ALIE distortion, and all compared methods use median-based defenses to filter the gradients. Aspis+ demonstrates an advantage of at least 15% compared with other algorithms (*cf.* $q = 2$). For $K = 25$, we tried a weaker distortion than ALIE, i.e., the constant attack paired with *signSGD*-based defenses [26]. In signSGD, the PS will output the majority of the gradients' signs for each dimension. Following the advice of [16], we pair this defense with the stronger constant attack as sign flips (*e.g.,* reversed gradient) are unlikely to affect the gradient's distribution. Aspis+ with median still enjoys an accuracy improvement of at least 20% for $q = 7$ and a larger one for $q = 9$. The results are in Figure 11; in this figure, the DETOX accuracy is an average of two experiments using two different random seeds.

VIII. Conclusions and Future Work

In this work, we have presented Aspis and Aspis+, two Byzantine-resilient distributed schemes that use redundancy and robust aggregation in novel ways to detect failures of the workers. Our theoretical analysis and numerical experiments clearly indicate their superior performance compared to state-of-the-art. Our experiments show that these methods require increased computation and communication time as compared to prior work, e.g., note that each worker has to transmit l gradients instead of 1 in related work [16], [17] (see Appendix Section IX-B4 for details). We emphasize, however, that our schemes converge to high accuracy in our experiments, while other methods remain at much lower accuracy values regardless of how long the algorithm runs for.

Our experiments involve clusters of up to 25 workers. As we scale Aspis to more workers, the total number of files and the computation load l of each worker will also scale; this increases the memory needed to store the gradients during aggregation. For complex neural networks, the memory to store the model and the intermediate gradient computations is by far the most memory-consuming aspect of the algorithm. For these reasons, Aspis is mostly suitable for training large data sets using fairly compact models that do not require too much memory. Aspis+, on the other hand, is a good fit for clusters that suffer from non-adversarial failures that can lead to inaccurate gradients. Finally, utilizing GPUs and

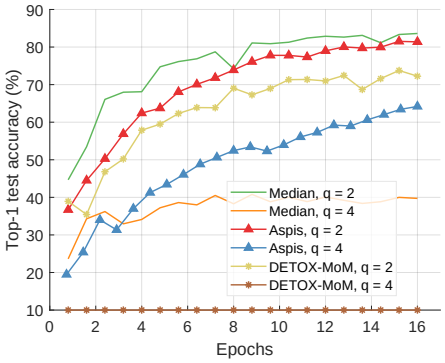


Fig. 6: *FoE* distortion under optimal attack scenarios, ATT-2 for Aspis, CIFAR-10, $K = 15$, median-based defenses.

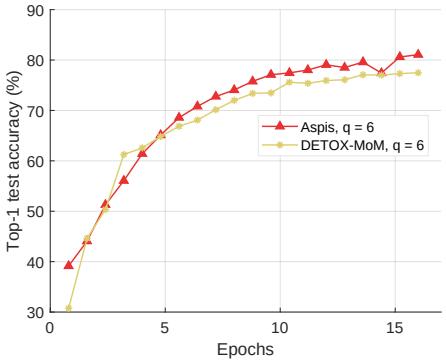


Fig. 7: *Reversed gradient* distortion under weak attack scenarios, ATT-1 for Aspis, CIFAR-10, $K = 15$, median-based defenses.

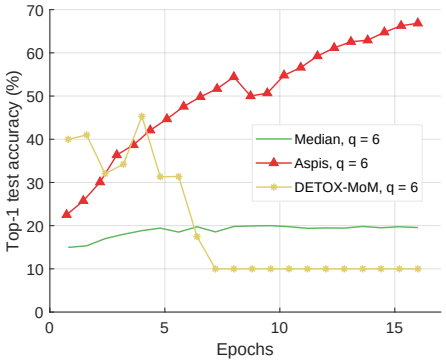
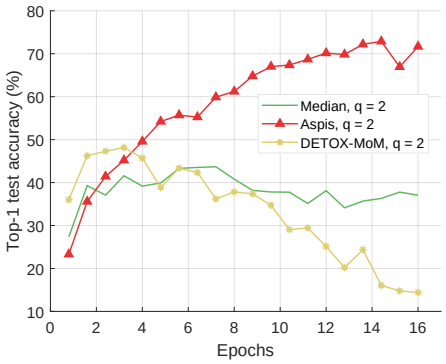
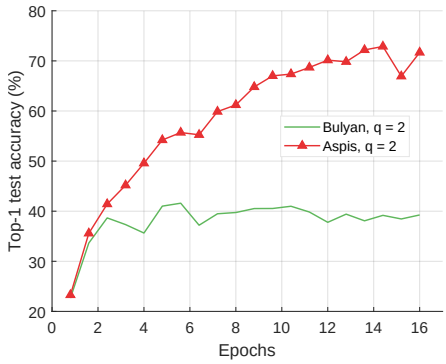


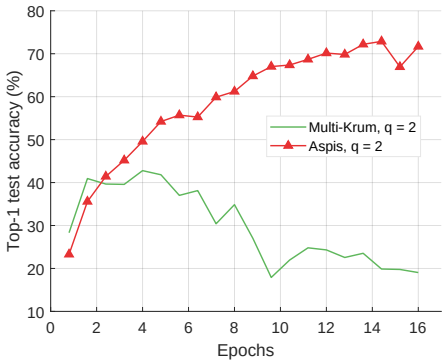
Fig. 8: *ALIE* distortion under weak attack scenarios, ATT-1 for Aspis, CIFAR-10, $K = 15$, median-based defenses.



(a) Median-based defenses.



(b) *Bulyan*-based defenses.



(c) *Multi-Krum*-based defenses.

Fig. 9: *ALIE* distortion under optimal attack scenarios, ATT-2 for Aspis, CIFAR-10, $K = 21$.

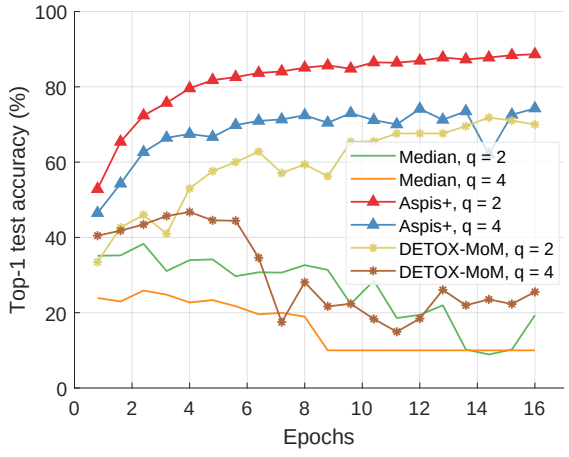


Fig. 10: *ALIE* distortion and random Byzantines, $K = 15$ (median-based defenses). ATT-3 used on Aspis+.

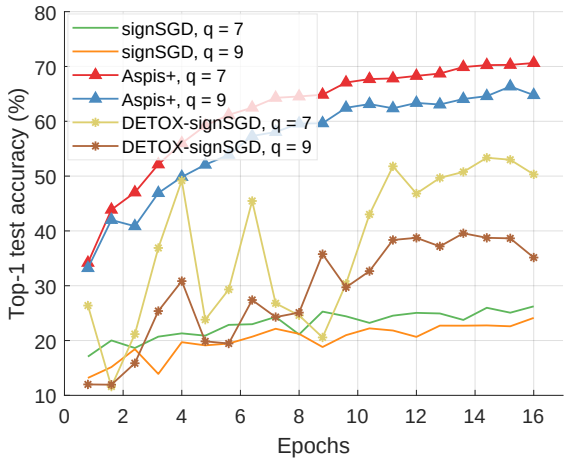


Fig. 11: *Constant* distortion and random Byzantines, $K = 25$ (*signSGD*-based defenses). ATT-3 used on Aspis+.

communication-related algorithmic improvements are worth exploring to reduce the time overhead.

IX. Appendix

A. Proof of Theorem 1

With X_j given in (5), assuming $q \geq r'$, the number of distorted files is upper bounded by

$$|\cup_{j=r'}^{r} X_j| \leq \sum_{j=r'}^{r} |X_j| \text{ (by the union bound).} \qquad (12)$$

TABLE II: Main notation of the paper.

Symbol	Meaning
K	number of workers
q	number of adversaries
r	redundancy (number of workers each file is assigned to)
b	batch size
B_t	samples of batch of t-th iteration
f	number of files (alternatively called *groups* or *tasks*)
U_j	j-th worker
l	computation load (number of files per worker)
$\mathcal{N}^w(U_j)$	set of files of worker U_j
$\mathcal{N}^f(B_{t,i})$	set of workers assigned to file $B_{t,i}$
$\mathbf{g}_{t,i}$	true gradient of file $B_{t,i}$ with respect to $\mathbf{w}$
$\hat{\mathbf{g}}_{t,i}^{(j)}$	returned gradient of U_j for file $B_{t,i}$ with respect to $\mathbf{w}$
$\mathbf{m}_i$	majority gradient for file $B_{t,i}$
$\mathcal{U}$	worker set $\{U_1, U_2, ..., U_K\}$
$\mathbf{G}_{task}$	graph used to encode the task assignments to workers
$\mathbf{G}_t$	graph indicating the agreements of pairs of workers in all of their common gradient tasks in t-th iteration
A	set of adversaries
$M_{\mathbf{G}}$	maximum clique in $\mathbf{G}$
$c^{(q)}$	number of distorted gradients after detection and aggregation
$c_{\max}^{(q)}$	maximum number of distorted gradients after detection and aggregation (worst-case)
D_i	disagreement set (of workers) for i^{th} adversary
r'	$(r+1)/2$, i.e., the minimum number of distorted copies needed to corrupt the majority vote for a file
ϵ	$c^{(q)}/f$, i.e., the fraction of distorted gradients after detection and aggregation
X_j	subset of files where the set of active adversaries is of size j

For that, recall that $r' = r(r+1)/2$ and that an adversarial majority of at least r' distorted computations for a file is needed to corrupt that particular file. Note that X_j consists of those files where the active adversaries A' are of size j; these can be chosen in $\binom{q}{j}$ ways. The remaining workers in the file belong to $\cap_{i\in A'} D_i$ where $|\cap_{i\in A'} D_i| \leq q$. Thus, the remaining workers can be chosen in at most $\binom{q}{r-j}$ ways. It follows that

$$|X_j| \leq \binom{q}{j}\binom{q}{r-j}. \tag{13}$$

Therefore,

$$\begin{aligned} c_{\max}^{(q)} &\leq \binom{q}{r'}\binom{q}{r-r'} + \binom{q}{r'+1}\binom{q}{r-(r'+1)} \\ &\quad + \cdots \\ &\quad + \binom{q}{r-1}\binom{q}{r-(r-1)} + \binom{q}{r} && (14) \\ &= \sum_{i=r'}^{q}\binom{q}{i}\binom{q}{r-i} && (15) \\ &= \sum_{i=0}^{q}\binom{q}{i}\binom{q}{r-i} - \sum_{i=0}^{r'-1}\binom{q}{i}\binom{q}{r-i} && (16) \\ &= \frac{1}{2}\binom{2q}{r}. && (17) \end{aligned}$$

Eq. (15) follows from the convention that $\binom{n}{k} = 0$ when $k > n$ or $k < 0$. Eq. (17) follows from Eq. (16) using the following observations

- $\sum_{i=0}^{q}\binom{q}{i}\binom{q}{r-i} = \sum_{i=0}^{r}\binom{q}{i}\binom{q}{r-i} = \binom{2q}{r}$ in which the first equality is straightforward to show by taking all possible cases: $q < r$, $q = r$ and $q > r$.
- By symmetry, $\sum_{i=0}^{r'-1}\binom{q}{i}\binom{q}{r-i} = \sum_{i=r'}^{q}\binom{q}{i}\binom{q}{r-i} = \frac{1}{2}\binom{2q}{r}$.

The upper bound in Eq. (14) is met with equality when all adversaries choose the same disagreement set, which is a q-sized subset of the honest workers, i.e., $D_i = D \subset H$ for $i = 1, \ldots, q$. In this case, it can be seen that the sets $X_j, j = r', \ldots, r$ are disjoint so that (12) is met with equality. Moreover, (13) is also an equality. This finally implies that (14) is also an equality, i.e., this choice of disagreement sets saturates the upper bound.

It can also be seen that in this case, the adversarial strategy yields a graph $\mathbf{G}$ with multiple maximum cliques. To see this, we note that the adversaries in A agree with all the computed gradients in $H \setminus D$. Thus, they form a clique of $M_{\mathbf{G}}^{(1)}$ of size $K-q$ in $\mathbf{G}$. Furthermore, the honest workers in H form another clique $M_{\mathbf{G}}^{(2)}$, which is also of size $K-q$. Thus, the detection algorithm cannot select one over the other, and the adversaries will evade detection; and the fallback robust aggregation strategy will apply.

B. Experiment Setup Details

1) Cluster Setup: We used clusters of $K = 15$, 21, and 25 workers arranged in various setups within Amazon EC2. Initially, we used a PS of type `i3.16xlarge` and several workers of type `c5.4xlarge` to set up a distributed cluster; for the experiments, we adapted GPUs, `g3s.xlarge` instances were used. However, purely distributed implementations require training data to be transmitted from the PS to every single machine, based on our current implementation; an alternative approach one can follow is to set up shared storage space accessible by all machines to store the training data. Also, some instances were automatically terminated by AWS per the AWS *spot instance* policy limitations;[5] this incurred some delays in resuming the experiments that were stopped. In order to facilitate our evaluation and avoid these issues, we decided to simulate the PS and the workers for the rest of the experiments on a single instance either of type `x1.16xlarge` or `i3.16xlarge`. We emphasize that the choice of the EC2 setup does not affect any of the numerical results in this paper since, in all cases, we used a single virtual machine image with the same dependencies. Handling of the GPU floating-point precision errors has been discussed in Supplement Section X-B.

2) Data Set Preprocessing and Hyperparameter Tuning: The CIFAR-10 images have been normalized using standard mean and standard deviation values for the data set. The value used for momentum (for gradient descent) was set to 0.9, and we trained for 16 epochs in all experiments. The number of epochs is precisely the invariant we maintain across all experiments, i.e., all schemes process the training data the same number of times. The batch size and the learning rate are chosen independently for each method; the number of iterations is adjusted accordingly to account for the number of epochs. For Section VII-B, we followed the advice of

[5]https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-interruptions.html

TABLE III: Distortion fraction of optimal and weak attacks for $(K, f, l, r) = (15, 455, 91, 3)$ and comparison.

q	ϵ^{Aspis}_{ATT-2}	$\epsilon^{Baseline}$	ϵ^{DETOX}	$\epsilon^{ByzShield}$
2	0.004	0.133	0.2	0.04
3	0.022	0.2	0.2	0.12
4	0.062	0.267	0.4	0.2
5	0.132	0.333	0.4	0.32
6	0.242	0.4	0.6	0.48
7	0.4	0.467	0.6	0.56

III(a) Optimal attacks.

q	ϵ^{Aspis}_{ATT-1}	$\epsilon^{Baseline}$	ϵ^{DETOX}
2	0.002	0.133	0
3	0.002	0.2	0
4	0.009	0.267	0
5	0.022	0.333	0
6	0.044	0.4	0.2
7	0.077	0.467	0.4

III(b) Weak attacks.

TABLE IV: Distortion fraction of optimal and weak attacks for $(K, f, l, r) = (21, 1330, 190, 3)$ and comparison.

q	ϵ^{Aspis}_{ATT-2}	$\epsilon^{Baseline}$	ϵ^{DETOX}	$\epsilon^{ByzShield}$
2	0.002	0.095	0.143	0.02
3	0.008	0.143	0.143	0.06
4	0.021	0.19	0.286	0.1
5	0.045	0.238	0.286	0.16
6	0.083	0.286	0.429	0.24
7	0.137	0.333	0.429	0.33
8	0.211	0.381	0.571	0.43
9	0.307	0.429	0.571	0.51
10	0.429	0.476	0.714	0.59

IV(a) Optimal attacks.

q	ϵ^{Aspis}_{ATT-1}	$\epsilon^{Baseline}$	ϵ^{DETOX}
2	0.001	0.095	0
3	0.001	0.143	0
4	0.003	0.19	0
5	0.008	0.238	0
6	0.015	0.286	0
7	0.026	0.333	0
8	0.042	0.381	0.143
9	0.063	0.429	0.286
10	0.09	0.476	0.429

IV(b) Weak attacks.

TABLE V: Distortion fraction of optimal and weak attacks for $(K, f, l, r) = (24, 2024, 253, 3)$ and comparison.

q	ϵ^{Aspis}_{ATT-2}	$\epsilon^{Baseline}$	ϵ^{DETOX}	$\epsilon^{ByzShield}$
2	0.001	0.083	0.125	0.031
3	0.005	0.125	0.125	0.063
4	0.014	0.167	0.25	0.125
5	0.03	0.208	0.25	0.188
6	0.054	0.25	0.375	0.281
7	0.09	0.292	0.375	0.375
8	0.138	0.333	0.5	0.5
9	0.202	0.375	0.5	0.5
10	0.282	0.417	0.625	0.531
11	0.38	0.458	0.625	0.625

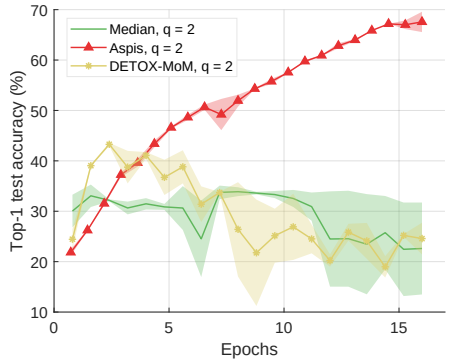
V(a) Optimal attacks.

q	ϵ^{Aspis}_{ATT-1}	$\epsilon^{Baseline}$	ϵ^{DETOX}
2	0	0.083	0
3	0	0.125	0
4	0.002	0.167	0
5	0.005	0.208	0
6	0.01	0.25	0
7	0.017	0.292	0
8	0.028	0.333	0
9	0.042	0.375	0.125
10	0.059	0.417	0.25
11	0.082	0.458	0.375

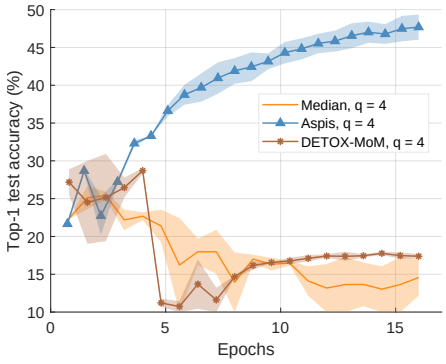
V(b) Weak attacks.

the authors of DETOX and chose $(K, b) = (15, 480)$ and $(K, b) = (21, 672)$ for the DETOX and baseline schemes. For Aspis, we used $(K, b) = (15, 14560)$ (32 samples per file) and $(K, b) = (21, 3990)$ (3 samples per file) for the ALIE experiments and $b = 1365$ (3 samples per file) for the remaining experiments except for the FoE optimal attack $q = 4$ (*cf.* Figure 6) for which $b = 14560$ performed better. In Section VII-C, we used $(K, b) = (15, 480)$ and $(K, b) = (25, 800)$ for DETOX as well as for baseline schemes while for Aspis+ we used $(K, b) = (15, 770)$ for the ALIE experiments and $(K, b) = (25, 1800)$ for the constant attack experiments. In Supplement Table VI, a learning rate schedule is denoted by (x, y); this notation signifies the fact that we start with a rate equal to x, and every z iterations, we set the rate equal to $x \times y^{t/z}$, where t is the index of the current iteration and z is set to be the number of iterations occurring between two consecutive checkpoints in which we store the model (points in the accuracy figures). We will also index the schemes in order of appearance in the corresponding figure's legend. Experiments that appear in multiple figures are not repeated in Supplement Table VI (we ran those training processes once). In order to pick the optimal hyperparameters for each scheme, we performed an extensive grid search involving different combinations of (x, y). In particular, the values of x we tested are 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, and 0.0003, and for y we tried 1, 0.975, 0.95, 0.7 and 0.5. For each method, we ran 3 epochs for each such combination and chose the one which was giving the lowest value of average cross-entropy loss (principal criterion) and the highest value of top-1 accuracy (secondary criterion).

3) Error Bars: In order to examine whether the choice of the random seed affects the accuracy of the trained model, we have performed the experiments of Section VII-B for the ALIE distortion for two different seeds for the values $q = 2, 4$ for

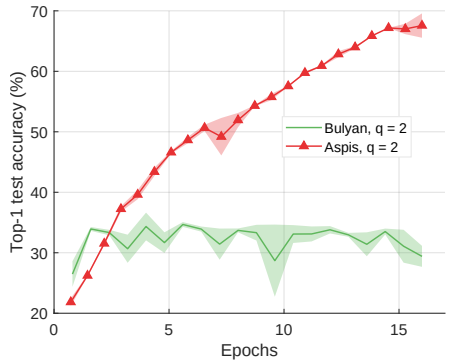


(a) $q = 2$ adversaries.

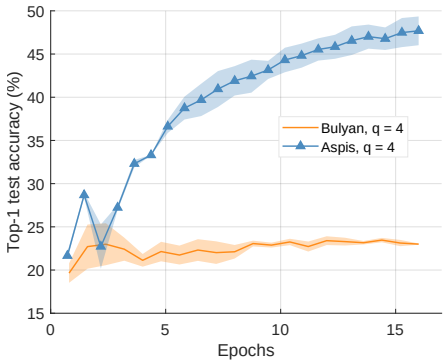


(b) $q = 4$ adversaries.

Fig. 12: *ALIE* optimal attack and median-based defenses (CIFAR-10), $K = 15$ with different random seeds, ATT-2 (Aspis).



(a) $q = 2$ adversaries.



(b) $q = 4$ adversaries.

Fig. 13: *ALIE* optimal attack and *Bulyan*-based defenses (CIFAR-10), $K = 15$ with different random seeds, ATT-2 (Aspis).

every scheme; we used 428 and 50 as random seeds. These tests have been performed for the case of $K = 15$ workers. In Figure 12a, for a given method, we report the minimum accuracy, the maximum accuracy, and their average for each evaluation point. We repeat the same process in Figures 13a and Supplement Figure 14a when comparing with Bulyan and Multi-Krum, respectively. The corresponding experiments for $q = 4$ are shown in Figures 12b, 13b, and Supplement Figure 14b.

Given the fact that these experiments take a significant amount of time and that they are computationally expensive, we chose to perform this consistency check for a subset of our experiments. Nevertheless, these results indicate that prior schemes [8], [16], [24] are very sensitive to the choice of the random seed and demonstrate an unstable behavior in terms of convergence. In all of these cases, the achieved value of accuracy at the end of the 16 epochs of training is small compared to Aspis. On the other hand, the accuracy results for Aspis are almost identical for both choices of the random seed.

4) Computation and Communication Overhead: Our schemes provide robustness under powerful attacks and sophisticated distortion methods at the expense of increased computation and communication time. Note that each worker has to perform l forward/backward propagation computations and transmit l gradients per iteration. In related baseline [12], [24] and redundancy-based methods [16], [17], each worker is responsible for a single such computation. Experimentally, we have observed that Aspis needs up to $5\times$ overall training time compared to other schemes to complete the same number of training epochs. We emphasize that the training time incurred by each scheme depends on a wide range of parameters, including the utilized defense, the batch size, and the number of iterations, and can vary significantly. Our implementation supports GPUs, and we used NVIDIA CUDA [39] for some experiments to alleviate a significant part of the overhead; however, a detailed time cost analysis is not an objective of our current work. Communication-related algorithmic improvements are also worth exploring. Finally, our implementation natively supports resuming from a checkpoint (trained model) and hence, when new data becomes available, we can only use that data to perform more training epochs.

5) Software: Our implementation of the Aspis and Aspis+ algorithms used for the experiments builds on ByzShield's [14] PyTorch skeleton and has been provided along with dependency information and instructions [6]. The implementation of ByzShield is available at [46] and uses the standard Github license. We utilized the NetworkX package [47] for the clique-finding; its license is 3-clause BSD. The CIFAR-10 data set [32] comes with the MIT license; we have cited its technical report, as required.

REFERENCES

[1] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner,

[6] https://github.com/kkonstantinidis/Aspis

L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, December 2019, pp. 8024–8035.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, November 2016, pp. 265–283.

[3] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," December 2015. [Online]. Available: https://arxiv.org/abs/1512.01274

[4] F. Seide and A. Agarwal, "CNTK: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2016, p. 2135.

[5] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, June 2014, pp. 361—372.

[6] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019, pp. 1211–1220.

[7] N. Gupta and N. H. Vaidya, "Byzantine fault-tolerant parallelized stochastic gradient descent for linear regression," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, September 2019, pp. 415–420.

[8] G. Damaskinos, E. M. El Mhamdi, R. Guerraoui, A. H. A. Guirguis, and S. L. A. Rouault, "Aggregathor: Byzantine machine learning via robust gradient aggregation," in *Conference on Systems and Machine Learning (SysML) 2019*, March 2019, p. 19.

[9] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Defending against saddle point attack in Byzantine-robust distributed learning," in *Proceedings of the 36th International Conference on Machine Learning*, June 2019, pp. 7074–7084.

[10] ——, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proceedings of the 35th International Conference on Machine Learning*, July 2018, pp. 5650–5659.

[11] C. Xie, O. Koyejo, and I. Gupta, "Generalized Byzantine-tolerant SGD," March 2018. [Online]. Available: https://arxiv.org/abs/1802.10116

[12] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, December 2017, pp. 119–129.

[13] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 2, December 2017.

[14] K. Konstantinidis and A. Ramamoorthy, "ByzShield: An efficient and robust system for distributed training," in *Machine Learning and Systems 3 (MLSys 2021)*, April 2021.

[15] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," April 2019. [Online]. Available: https://arxiv.org/abs/1806.00939

[16] S. Rajput, H. Wang, Z. Charles, and D. Papailiopoulos, "DETOX: A redundancy-based framework for faster and more robust gradient aggregation," in *Advances in Neural Information Processing Systems*, December 2019, pp. 10 320–10 330.

[17] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "DRACO: Byzantine-resilient distributed training via redundant gradients," in *Proceedings of the 35th International Conference on Machine Learning*, July 2018, pp. 903–912.

[18] D. Data, L. Song, and S. Diggavi, "Data encoding for Byzantine-resilient distributed gradient descent," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, October 2018, pp. 863–870.

[19] J. Regatti, H. Chen, and A. Gupta, "ByGARS: Byzantine SGD with arbitrary number of attackers," December 2020. [Online]. Available: https://arxiv.org/abs/2006.13421

[20] C. Xie, S. Koyejo, and I. Gupta, "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance," in *Proceedings of the 36th International Conference on Machine Learning*, June 2019, pp. 6893–6901.

[21] D. Alistarh, Z. Allen-Zhu, and J. Li, "Byzantine stochastic gradient descent," in *Advances in Neural Information Processing Systems*, December 2018.

[22] K. Konstantinidis and A. Ramamoorthy, "Aspis: A robust detection system for distributed learning," January 2022. [Online]. Available: https://arxiv.org/abs/2108.02416

[23] G. Baruch, M. Baruch, and Y. Goldberg, "A Little Is Enough: Circumventing defenses for distributed learning," in *Advances in Neural Information Processing Systems*, December 2019, pp. 8635–8645.

[24] E. M. El Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in Byzantium," in *Proceedings of the 35th International Conference on Machine Learning*, July 2018, pp. 3521–3530.

[25] S. Shen, S. Tople, and P. Saxena, "Auror: Defending against poisoning attacks in collaborative deep learning systems," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, December 2016, pp. 508—-519.

[26] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "signSGD with majority vote is communication efficient and fault tolerant," February 2019. [Online]. Available: https://arxiv.org/abs/1810.05291

[27] J. So, B. Güler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2168–2181, July 2021.

[28] R. Jin, Y. Huang, X. He, H. Dai, and T. Wu, "Stochastic-sign SGD for federated learning with theoretical guarantees," September 2021. [Online]. Available: https://arxiv.org/abs/2002.10940

[29] N. Raviv, R. Tandon, A. Dimakis, and I. Tamo, "Gradient coding from cyclic MDS codes and expander graphs," in *Proceedings of the 35th International Conference on Machine Learning*, July 2018, pp. 4302–4310.

[30] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning*, August 2017, pp. 3368–3376.

[31] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, May 2018.

[32] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[33] D. R. Stinson, *Combinatorial Designs: Constructions and Analysis*. New York: Springer, 2004.

[34] R. M. Karp, *Reducibility among Combinatorial Problems*. Boston, MA: Springer US, 1972.

[35] F. Cazals and C. Karande, "A note on the problem of reporting maximal cliques," *Theoretical Computer Science*, vol. 407, no. 1, pp. 564–568, November 2008.

[36] T. Etsuji, T. Akira, and T. Haruhisa, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science*, vol. 363, no. 1, pp. 28–42, October 2006.

[37] J. Robson, "Algorithms for maximum independent sets," *Journal of Algorithms*, vol. 7, no. 3, pp. 425–440, September 1986.

[38] R. E. Tarjan and A. E. Trojanowski, "Finding a maximum independent set," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 537–546, 1977.

[39] "NVIDIA CUDA toolkit," September 2022. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[40] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," April 2018. [Online]. Available: https://arxiv.org/abs/1804.07612

[41] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," September 2017. [Online]. Available: https://arxiv.org/abs/1708.03888

[42] J. Geiping, M. Goldblum, P. E. Pope, M. Moeller, and T. Goldstein, "Stochastic training is not necessary for generalization," April 2022. [Online]. Available: https://arxiv.org/abs/2109.14119

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[44] C. Xie, O. Koyejo, and I. Gupta, "Fall of Empires: Breaking byzantine-tolerant sgd by inner product manipulation," in *35th Conference on Uncertainty in Artificial Intelligence, UAI 2019*, July 2019, pp. 6893–6901.

[45] S. Minsker, "Geometric median and robust estimation in Banach spaces," *Bernoulli*, vol. 21, no. 4, pp. 2308–2335, November 2015.

[46] "Repository of ByzShield implementation," August 2022. [Online]. Available: https://github.com/kkonstantinidis/ByzShield

[47] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, August 2008, pp. 11–15.

[48] R. S. Boyer and J. S. Moore, *MJRTY—A Fast Majority Vote Algorithm*. Dordrecht: Springer Netherlands, 1991.

X. Supplement

A. Asymptotic Complexity

If the gradient computation has linear complexity (assuming $\mathcal{O}(1)$ cost for the gradient computation with respect to one model parameter) and since each worker is assigned to l files of b/f samples each, the gradient computation cost at the worker level is $\mathcal{O}((lb/f)d)$ (K such computations in parallel). In our schemes, however, b is a constant multiple of f, and in general $r < l$ ($l = \binom{K-1}{r-1}$ for Aspis, while $r = 3$ is a typical redundancy value used in literature as well as in Aspis+); hence, the complexity becomes $\mathcal{O}(ld)$ which is similar to other redundancy-based schemes in [14], [16], [17]. The clique-finding problem that follows as part of Aspis detection is NP-complete. However, our experimental evidence suggests that this computation takes an infinitesimal fraction of the execution time for the kind of graphs we construct. The NetworkX package [47], which we use for enumerating all maximal cliques, is based on the algorithm of [36] and has asymptotic complexity $\mathcal{O}(3^{K/3})$. We provide extensive simulations of the clique enumeration time under the Aspis file assignment for $K = 100$ and redundancy $r = 5$ (*cf.* Supplement Tables VII(a), VII(b) for the weak (ATT-1) and optimal (ATT-2) attack as introduced in Section IV, respectively). We emphasize that this value of K exceeds by far the typical values of K of prior work, and the number of servers would suffice for most challenging training tasks. Even in this case, the cost of enumerating all cliques is negligible. For this experiment, we used an EC2 instance of type `i3.16xlarge`. The complexity of robust aggregation varies significantly depending on the operator. For example, majority voting can be done in time, which scales linearly with the number of votes using *MJRTY* proposed in [48]. In our case, this is $\mathcal{O}(Kd)$ as the PS needs to use the d-dimensional input from all K machines. Krum [12], Multi-Krum [12] and Bulyan [24] are applied to all K workers by default and require $\mathcal{O}(K^2(d + \log K))$.

B. Floating-Point Precision and Gradient Equality Check

A gradient equality check is needed to determine whether two gradient vectors, e.g., $\mathbf{a}$ and $\mathbf{b}$, are equal for our majority voting procedure to work. This check can be performed on an element-by-element basis or using the norm of the difference. There are two distinct cases we have considered:

- *Case 1: Execution on CPUs*: If we use the CPUs of the workers to compute the gradients, we have observed that two "honest" gradients, $\mathbf{a}$ and $\mathbf{b}$, will always be exactly equal to each other element-wise. In this case, we use the `numpy.array_equal` function for all equality checks. If one of $\mathbf{a}$, $\mathbf{b}$ is corrupted and the other one is "honest," the program will effectively flag this as a disagreement between the corresponding workers.
- *Case 2: Execution on GPUs*: Most deep learning libraries [1], [2] provide non-deterministic back-propagation for the sake of faster and more efficient computations. In our implementation, we use NVIDIA CUDA [39]; hence, two "honest" float (e.g., `numpy.float_32`) gradients $\mathbf{a}$ and $\mathbf{b}$ computed by two different GPUs will not be exactly equal to each other. However, the floating-point precision errors were less than 10^{-6} in all of our experiments. In this case, we decide that the two workers agree with each other if the following criterion is satisfied for a small tolerance value of 10^{-5}

$$\frac{\|\mathbf{a}-\mathbf{b}\|_2}{\max\{\|\mathbf{a}\|_2, \|\mathbf{b}\|_2\}} \leq 10^{-5}.$$

On the other hand, if one of $\mathbf{a}$, $\mathbf{b}$ is distorted even by the most sophisticated inner manipulation attack ALIE [23], then $\frac{\|\mathbf{a}-\mathbf{b}\|_2}{\max\{\|\mathbf{a}\|_2, \|\mathbf{b}\|_2\}}$ is at least five orders of magnitude larger and typically ranges in $[1, 100]$.

In both cases, we have an integrity check in place to throw an exception if two "honest" gradients for the same task violate this criterion. We have not observed any violation of this in any of our exhaustive experiments.

TABLE VI: Parameters used for training.

Figure	Schemes	Learning rate schedule
4a	1,2,5,6	$(0.01, 0.7)$
4a	3,4	$(0.1, 0.95)$
4b	1	$(0.001, 0.95)$
4c	1,2	$(0.01, 0.7)$
5a	1,2	$(0.1, 0.7)$
5a	3,4	$(0.1, 0.95)$
5a	5,6	$(0.01, 0.7)$
5b	1	$(0.1, 0.7)$
5c	1,2	$(0.01, 0.975)$
6	1,2	$(0.1, 0.7)$
6	3,4	$(0.1, 0.95)$
6	5,6	$(0.01, 0.95)$
7	1	$(0.1, 0.95)$
7	2	$(0.01, 0.7)$
8	1	$(0.01, 0.7)$
8	2	$(0.1, 0.95)$
8	3	$(0.01, 0.7)$
9a	1,2	$(0.01, 0.7)$
9a	3	$(0.1, 0.95)$
9b	2	$(0.01, 0.7)$
9c	2	$(0.01, 0.95)$
10	1,2	$(0.01, 0.7)$
10	3,4	$(0.01, 0.975)$
10	5,6	$(0.1, 0.975)$
11	1,2,3,4	$(0.0003, 0.7)$
11	5,6	$(0.3, 0.975)$

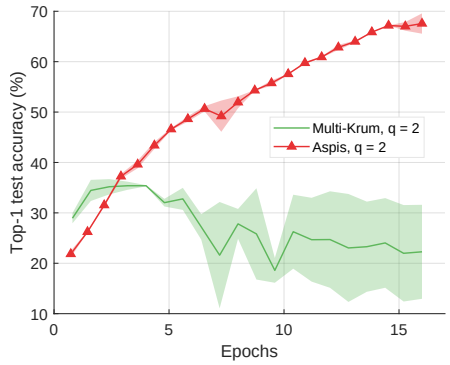
TABLE VII: Clique enumeration time in Aspis graph of $K = 100$ vertices and redundancy $r = 5$.

q	Time (milliseconds)
5	9
15	7
25	5
35	5
45	5

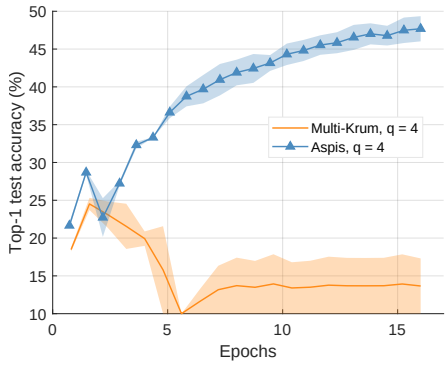
VII(a) Adversaries carry out weak attack ATT-1.

q	Time (milliseconds)
5	11
15	11
25	9
35	8
45	6

VII(b) Adversaries carry out optimal attack ATT-2.



(a) $q = 2$ adversaries.



(b) $q = 4$ adversaries.

Fig. 14: *ALIE* optimal attack and *Multi-Krum*-based defenses (CIFAR-10), $K = 15$ with different random seeds, ATT-2 (Aspis).