Zhixiong Yang, Arpita Gang, and Waheed U. Bajwa

Adversary-Resilient Distributed and Decentralized Statistical Inference and Machine Learning

An overview of recent advances under the Byzantine threat model

©ISTOCKPHOTO.COM/HAMSTER3D

Statistical inference and machine-learning algorithms have traditionally been developed for data available at a single location. Unlike this centralized setting, modern data sets are increasingly being distributed across multiple physical entities (sensors, devices, machines, data centers, and so on) for a multitude of reasons that range from storage, memory, and computational constraints to privacy concerns and engineering needs. This has necessitated the development of inference and learning algorithms capable of operating on noncolocated data. For this article, we divide such algorithms into two broad categories, namely, distributed algorithms and decentralized algorithms (see "Is It Distributed or Is It Decentralized?").

Distributed algorithms

Distributed algorithms correspond to setups in which the data-bearing entities (henceforth referred to as *nodes*) require some form of coordination through a specially designated entity in the system to generate the final result. Depending on the application, this special entity is referred to as *master node*, *central server*, *parameter server*, *fusion center*, and so on in the literature. Although distributed setups can take a number of forms, this exposition mostly revolves around the so-called master–worker distributed architecture in which data-bearing nodes communicate only with a single master node, which is tasked with generating the final result. Among other applications, such distributed architectures arise in the context of parallel computing, where the focus is computational speedups and/or overcoming memory/storage bottlenecks, and federated systems, where "raw" data collected by individual nodes cannot be shared with the master node due to either communication constraints (e.g., sensor networks) or privacy concerns (e.g., smartphone data).

Decentralized algorithms

Decentralized algorithms correspond to setups that lack central servers; instead, data-bearing nodes in a decentralized system are collectively tasked with generating the final result. In particular, individual nodes in a decentralized setup typically communicate among themselves over a network (often ad hoc) to reach a

Digital Object Identifier 10.1109/MSP.2020.2973345
Date of current version: 28 April 2020

 1053-5888/20©2020IEEE

Is It Distributed or Is It Decentralized?

Inference and learning from noncolocated data have been studied for decades in computer science, control, signal processing, and statistics. Both among and within these disciplines, however, there is no consensus on the use of the terms *distributed* and *decentralized*. Although many works share the definitions provided in this article, there are numerous authors who use these two terms interchangeably, while there are some other authors who reverse these definitions.

common solution (i.e., achieve consensus) at all nodes. Decentralized setups arise either out of the need to eliminate single points of failure in distributed setups or due to practical engineering constraints, as in the Internet of Things and autonomous systems.

The last few decades have witnessed the emergence of a plethora of applications that necessitate advances in inference and learning in distributed and decentralized setups. Indeed, distributed and decentralized statistical inference methods are increasingly being relied upon in urban traffic monitoring, environmental sensing, management of smart grids, distributed spectrum sensing, and homeland security, among other applications. Similarly, distributed and decentralized machine-learning algorithms are being utilized more in the context of networks of self-driving cars, the control of robot swarms, pattern recognition in large-scale data sets, and federated learning systems for health-care data. Collectively, these applications have resulted in the development of a huge body of work devoted to understanding the algorithmic and theoretical underpinnings of distributed and decentralized inference and learning. But, much of this work assumes a nonadversarial setting in which individual nodes, apart from occasional statistical failures, operate as intended within the algorithmic framework.

In recent years, however, cybersecurity threats from malicious nonstate actors and rogue entities and, the potentially disastrous consequences of these threats for the aforementioned applications, have forced practitioners and researchers to rethink the robustness of distributed and decentralized algorithms against adversarial attacks. As a result, we now have an abundance of algorithmic approaches that guarantee the robustness of distributed and/or decentralized inference and learning under different adversarial threat models (see, e.g., recent survey articles [1]–[3]).

Driven in part by the world's growing appetite for data-driven decision making, however, the securing of distributed/decentralized frameworks for inference and learning against adversarial threats remains a rapidly evolving research area. In this article, we provide an overview of some of the most recent developments in this area under the threat model of Byzantine attacks. This threat model, which subsumes the case of malfunctioning nodes, referred to as *Byzantine faults/failures*, is one of the hardest to safeguard against because it allows for the undetected takeover of nodes by the adversary. In particular, nodes affected under this threat model, termed *Byzantine nodes*, are assumed to have the potential to arbitrarily bias the outputs of the underlying algorithms by colluding among themselves, injecting false data and information into the distributed/decentralized system and so forth. This is in stark contrast to the relatively simpler threat model of crash faults, in which individual nodes in the distributed/decentralized system continue to operate as intended until a crash fault occurs, at which point the faulty node ceases to interact with the system (rather than potentially injecting false information into the system). We refer the reader to [4] and the references therein for further discussion on both the generality and the hardness of the Byzantine threat model in relation to the crash-fault model.

Since its introduction in 1982, the Byzantine threat model has been extensively studied in the context of statistical inference from noncolocated data [1], [2] (see "Origination of the Byzantine Threat Model"). However, its potential impact on more general decentralized consensus and distributed/decentralized machine-learning problems has only been recently studied. It is against this backdrop that this article summarizes recent developments in the Byzantine-resilient processing of noncolocated data, with a majority of the discussion focused on machine-learning problems in distributed and decentralized settings.

Adversary-resilient distributed processing of data

In the classic master–worker setting of distributed systems tasked with processing of noncolocated data, there is one central server, referred to as *master node*, *parameter server*, *fusion center*, and so on, which coordinates with M data-bearing nodes (sensors, smartphones, worker nodes, and so forth) for final decision making. Even though all nodes in this setting communicate with the server, they usually cannot communicate with each other [see Figure 1(a)]. The Byzantine threat model in this setting assumes that at most, b nodes in the system have been compromised. This parameter b, which typically corresponds to a crude upper bound on the exact number of Byzantine nodes, often plays an important role in both the analysis and performance of Byzantine-resilient algorithms. We summarize some of these algorithms and their theoretical guarantees in the following for statistical inference and machine-learning problems.

Distributed statistical inference

Statistical inference leverages data samples to draw conclusions about the underlying probability distribution(s) generating the data. Although statistical inference can take many forms, in this article, we limit our discussion to Byzantine-resilient distributed detection and distributed estimation.

Distributed detection

Distributed detection under both the Neyman–Pearson and Bayesian frameworks has a rich history. Given two hypotheses, $\mathcal{H}_0$ and $\mathcal{H}_1$, a typical distributed detection algorithm first involves each node taking a local decision in favor of either $\mathcal{H}_0$ or $\mathcal{H}_1$ based on its own data samples. The nodes then send their decisions to the central server, which applies a fusion rule to the local decisions to reach the final (global) decision. Unfortunately, distributed detection algorithms designed without the consideration of potential Byzantine failures break down in the presence of Byzantine nodes.

Despite the brittleness of traditional distributed detection techniques, the investigation of Byzantine-resilient distributed detection only took off in the last decade. A survey article [1] provides an overview of many of the resulting methods and asymptotically characterizes the respective critical fraction α^* of Byzantine nodes, defined as $\alpha^* := b/M$, at (or beyond) the point where the central server cannot do better than randomly guess for its final decision. An important

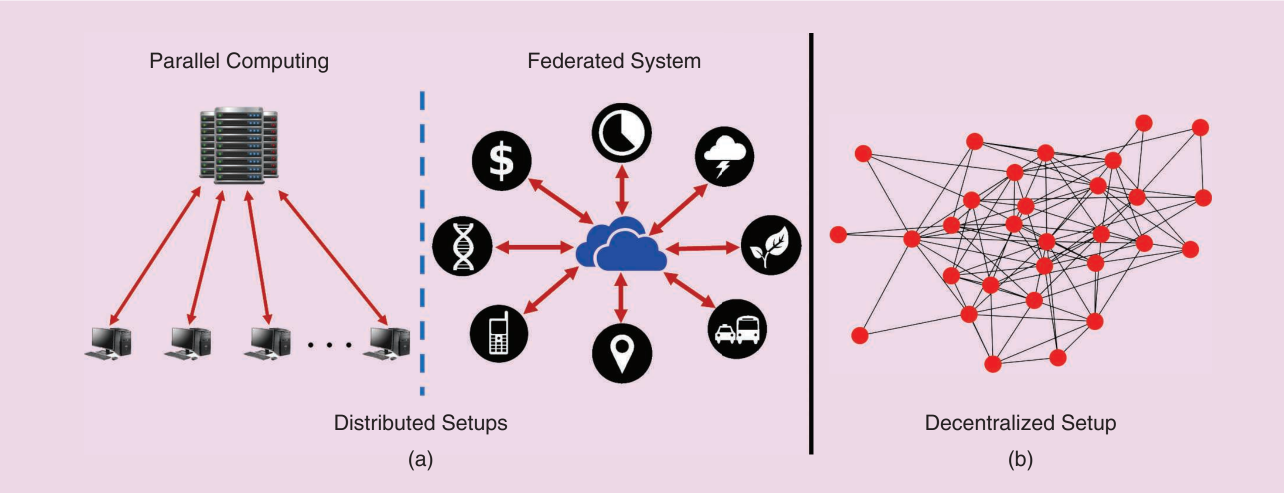

FIGURE 1. Inference and machine-learning algorithms involving noncolocated data can be broadly divided into the categories of (a) distributed algorithms and (b) decentralized algorithms. The former category includes master–worker distributed setups such as parallel computing and federated systems in which a single entity is tasked with generating the final result. The latter category deals with decentralized setups in which entities cooperate among themselves to produce the final result.

Origination of the Byzantine Threat Model

The threat model of Byzantine attacks/faults/failures in its most general form was introduced and analyzed in [5] within the context of the reliability of computer systems with potentially malfunctioning components. The overarching problem in [5] was abstracted as the Byzantine Generals Problem, in which several generals of the Byzantine army, some of whom are likely to be traitors, need to agree on attacking or retreating from an enemy city through the exchange of messages via a messenger (see, e.g., Figure S1). The authors reported two key results in [5] for this problem. First, they established the impossibility of safeguarding against Byzantine nodes (traitor generals) when the number of uncompromised nodes (loyal generals) is not more than two-thirds of the total number of nodes. The threat from General 3 in Figure S1, therefore, cannot be neutralized by any algorithm because the number of loyal generals is only two in this scenario. Second, the authors proposed two algorithms for complete and regular graphs that provably counter the actions of Byzantine nodes in the case of binary (e.g., attack/retreat) decisions as long as the number of uncompromised nodes exceeds two-thirds of the total number of nodes.

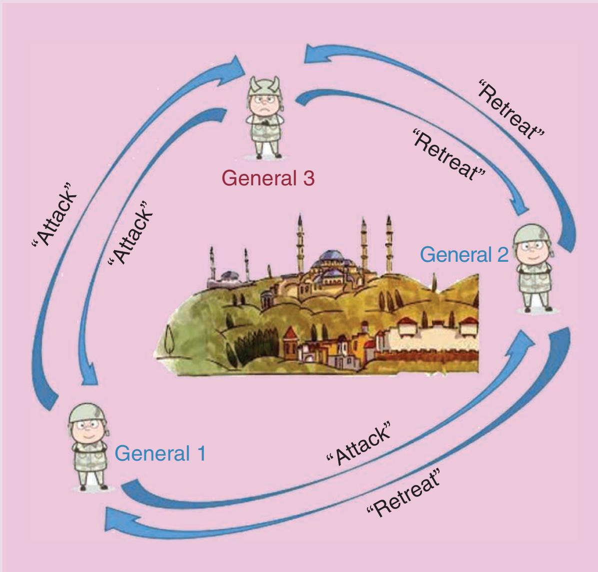

FIGURE S1. A Byzantine army led by three generals, one of whom (General 3) is a traitor, surrounding an enemy city. The loyal generals are trying to reach a consensus on the plan of action against the enemy, while the traitor is trying to mislead them.

insight from the earliest works on Byzantine-resilient distributed detection is that (asymptotically) α^* can be 1/2 or higher [1]; in contrast, recall that $\alpha^* = 1/3$ for the original Byzantine Generals Problem [5].

Since the appearance of [1] in 2013, a few other works on Byzantine-resilient distributed detection have appeared in the literature. It is shown in [6] that distributed detection can be more resilient to Byzantine failures in the case of a general Q-ary hypothesis testing problem, with the critical fraction given by $\alpha^* = (Q-1)/Q$. Byzantine-resilient distributed binary hypothesis testing is investigated for the first time under the Bayesian framework in [7], with the critical fraction also given by $\alpha^* = 1/2$ in this case. Finally, it is established in [8] that, under certain conditions, it is possible to have $\alpha^* = 1$ in the case of Bayesian distributed binary detection as long as each node is allowed to replicate its message to the central server through one other node in the system. Strictly speaking, however, this coordination among pairs of nodes leads to a distributed architecture that differs from the distributed master–worker (star topology) architecture of prior works. We conclude by noting that, a tree topology in which the central server sits at the root of the tree (depth 0) and nodes in the distributed system route their messages to the server through their parent nodes, is another distributed architecture that does not fall under the distributed master–worker setup. The Byzantine resilience of such architectures in distributed detection tasks is investigated in [9], with the critical fraction of Byzantine nodes defined in terms of the number of Byzantine nodes b_1 and the total number of nodes M_1 at depth 1 of the tree. We also refer the reader to Table 1 for a summary of all the results that have been discussed in this article in relation to Byzantine-resilient distributed detection.

Distributed estimation

Byzantine-resilient distributed estimation has received significant attention lately in the context of state estimation in cyberphysical systems. Many of the developments in this regard have been limited to linear models, with a typical observation model at any given time at node j expressed as $y_j = \mathbf{h}_j^T \mathbf{w} + \eta_j,\ j = 1, \ldots, M$, where $\mathbf{h}_j \in \mathbb{R}^d$ denotes a known vector, $\mathbf{w} \in \mathbb{R}^d$ is the unknown state vector that needs to be estimated at the central server, and η_j denotes the observation noise at node j. Traditionally, distributed estimation has involved the nodes transmitting y_j's (or their quantized versions) to the central server and the server estimating $\mathbf{w}$ using some variant of the following least-squares formulation [9] (or a maximum-likelihood one in the case of quantized transmissions):

$$\hat{\mathbf{w}} = \arg\min_{\tilde{\mathbf{w}}} \| \mathbf{y} - \mathbf{H}\tilde{\mathbf{w}} \|_2, \qquad (1)$$

where the matrix $\mathbf{H} \in \mathbb{R}^{M \times d}$ has $\mathbf{h}_j$s as its rows. The quality of such solutions are assessed in terms of the gap of their mean squared error (MSE), $\mathbb{E}[\| \hat{\mathbf{w}} - \mathbf{w} \|_2^2]$, from either the minimum MSE (MMSE) or the Cramer–Rao lower bound (CRLB) for unbiased estimators.

In the presence of Byzantine nodes, in which case the y_js corresponding to the b Byzantine nodes can be arbitrarily different from $\mathbf{h}_j^T \mathbf{w}$, the solutions of distributed estimation techniques based on (1) can be pushed far from the optimal (in terms of MMSE/CRLB) (see "Distributed Estimation and Byzantine Failures"). In this setting, one can again characterize the critical fraction α^* of Byzantine nodes at (or beyond) the point where the server can do no better than relying on prior knowledge about $\mathbf{w}$. It is, for instance, established in [6] that, under certain assumptions, it is possible to have $\alpha^* = (Q-1)/Q$ when the nodes utilize Q-ary quantization to transmit their data to the server. Several survey articles [1]–[3] provide additional discussion of works on Byzantine-resilient distributed estimation, many of which are based on the idea of either the detection of Byzantine attacks and/or the identification of individual Byzantine nodes as part of their mitigation strategy. Finally, while much of the focus in Byzantine-resilient distributed estimation has been on linear models, there are works such as [11] that focus on nonlinear models. Because there is a significant overlap between statistical estimation and machine-learning problems, we do not indulge in the discussion of such works; instead, we move toward our review of recent results on Byzantine-resilient distributed machine learning.

Distributed machine learning

A typical challenge in machine learning is to statistically minimize a function f, referred to as *loss function* or *risk function*, with respect to a candidate model $\mathbf{w} \in \mathbb{R}^d$ that describes the data, i.e.,

$$\min_{\mathbf{w}} \mathbb{E}_{\mathbf{z} \sim \mathcal{P}}[f(\mathbf{w}, \mathbf{z})], \qquad (2)$$

where $\mathbf{z}$ denotes data living in some Hilbert space that is drawn from an unknown probability distribution $\mathcal{P}$. To solve (2) without the knowledge of $\mathcal{P}$, distributed machine learning focuses on minimizing an empirical variant of $\mathbb{E}_{\mathbf{z} \sim \mathcal{P}}[f(\mathbf{w}, \mathbf{z})]$ using samples of $\mathbf{z}$, termed *training data*, distributed across different nodes. The resulting objective,

Table 1. A summary of recent results concerning Byzantine-resilient distributed detection.

Reference	Hypothesis Test	Detection Setup	Critical Fraction α^*	Distributed Architecture
Nadendla et al. [6]	Q-ary	Neyman–Pearson	$\alpha^* = \frac{Q-1}{Q}$	Star topology
Kailkhura et al. [7]	Binary	Bayesian	$\alpha^* = 0.5$	Star topology
Hashlamoun et al. [8]	Binary	Bayesian	Can have $\alpha^* = 1$	Quasi-star topology
Kailkhura et al. [9]	Binary	Neyman–Pearson	$b_1 = \left\lceil \frac{M_1}{2} \right\rceil$	Tree topology

aptly termed distributed *empirical risk minimization* (*ERM*), can be expressed as $\min_{\mathbf{w}}(1/M)\Sigma_{j=1}^{M} f(\mathbf{w}, \mathbf{Z}_j)$, with $\mathbf{Z}_j$ denoting the local training data at node j that comprises multiple samples drawn from $\mathcal{P}$ and $\mathbf{w}$ referred to as the *global* optimization variable. The model/variable $\mathbf{w}$ in distributed machine learning is stored at the central server, which iteratively updates it based on messages received from individual nodes and subsequently sends the updated $\mathbf{w}$ to all of the nodes. Each node, taking turns, performs some computation according to its local data and the received $\mathbf{w}$ and sends a message back to the server.

Distributed stochastic gradient descent

Similar to distributed statistical inference, Byzantine failures can lead to breakdowns of distributed machine-learning methods. Motivated in part by the widespread adoption of deep neural networks and variance-reduction techniques like minibatching by the practitioners, here, we mainly focus on the robustification of synchronous distributed stochastic gradient descent (SGD) against Byzantine attacks. A typical distributed SGD algorithm proceeds iteratively. In each iteration, nodes compute the gradient of the loss function on their local data with respect to the current model and send their gradients to the server. The server, in turn, takes the average of all the gradients and updates the global variable according to the averaged gradient. In a faultless environment, distributed SGD, with its proper choices of step size and batch size, has a linear convergence rate for strongly convex functions; however, a single Byzantine node can force the algorithm to converge to any model using a simple strategy.

Suppose, for instance, that the summation of gradients of faultless nodes is $\mathbf{g}$ and the Byzantine node wishes for the server to operate on an alternate gradient $\mathbf{g}'$. The Byzantine node can accomplish this by sending $M\mathbf{g}' - \mathbf{g}$ as its gradient to the server, thereby controlling the update step at the server. Of course, as simple as this strategy is, it is not an optimal one for the Byzantine node; indeed, a large M will result in a large gradient, which is likely to make the malicious message detectable. We refer the reader to [12]–[14] for more sophisticated strategies that can be employed by Byzantine nodes.

Recently, several algorithms have been put forth to safeguard distributed SGD against Byzantine failures. The central idea in all of these approaches that imparts Byzantine resilience to distributed SGD involves the use of a screening procedure at the server while it aggregates the local gradients. Using an appropriate screening rule, the screened aggregated gradient can be shown to be close to the true average gradient, thereby enabling the server to approximately solve the distributed ERM problem. Algorithm 1 provides a general framework for a Byzantine-resilient distributed SGD based on this idea of screening. Even though some algorithms alter this framework a little bit, the general idea remains the same. Note that if one were to remove the screening procedure in step 10 of the algorithm, it becomes vanilla distributed SGD.

Distributed Estimation and Byzantine Failures

We illustrate the impact of Byzantine nodes on distributed estimation through a simple example of 2D parameter estimation, in which the linear observation model corresponds to a line in the (x, y) plane, using $M = 8$ nodes. The observations at each node in this example correspond to $y_i = \mathbf{h}_i^T \mathbf{w} + \eta_i$ with $\mathbf{h}_i^T := [x_i\, 1]$ and the 2D vector $\mathbf{w}$ describing the slope and y-intercept of the line. It can be seen from Figure S2 that Byzantine failures of just two nodes return a least-squares solution that results in a model $\mathbf{h}_i^T \hat{\mathbf{w}}$ that significantly differs from the true model, $\mathbf{h}_i^T \mathbf{w}$.

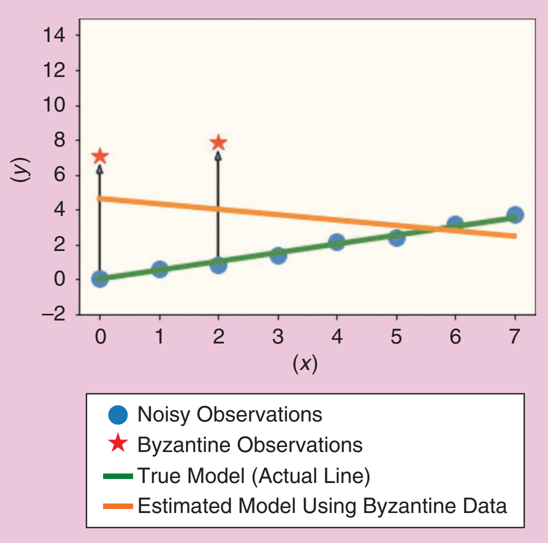

FIGURE S2. The impact of Byzantine failures on traditional least-squares estimation.

Algorithm 1: A general framework for Byzantine-resilient distributed SGD.

Step 1: **for** $t = 1, 2, \ldots$ **do**
Step 2: **Server:**
Step 3: Send the global optimization variable to all the nodes
Step 4: **Node:**
Step 5: Receive the global optimization variable from the server
Step 6: Calculate gradient with respect to the local training set
Step 7: Send the local gradient to the server
Step 8: **Server:**
Step 9: Receive local gradients from all nodes
Step 10: Screen the received gradients for Byzantine resilience and aggregate them
Step 11: Update the global variable by taking a gradient step using the aggregated gradient
Step 12: **end for**

The state of the art

In this section, we discuss the various screening (and aggregation) procedures adopted in different algorithms (see "Screening and Aggregation in Two Dimensions"). The algorithm introduced in [15], termed *robust distributed gradient descent*, uses two screening methods: coordinate-wise median (CM) and coordinate-wise trimmed mean (CTM). In CM, server aggregates the local gradients by taking the median in each dimension of the gradients. CTM, on the other hand, involves the server eliminating the smallest and the largest b values in each dimension of the gradients and coordinate-wise averaging the remaining values for aggregation. The GeoMed algorithm [16], in contrast, uses the geometric median of local gradients as the screening and aggregation rule. The Krum algorithm [17], on the other hand, finds the local gradient that has the smallest distance to its $M-b-2$ closest gradients and uses this gradient for the update step. There is also a variant of the Krum algorithm, termed *multi-Krum* [17], which finds $m \in \{1,\dots,M\}$ local gradients using the Krum principle and uses an average of these gradients for update.

The Bulyan algorithm [18], [19] is a two-stage algorithm. In the first stage, it recursively uses vector median methods such as geometric median and Krum to select $M-2b$ local gradients. In the second stage, it carries out a coordinate-wise operation on the $M-2b$ selected gradients in which $M-4b$ values in each coordinate are retained (and then averaged for aggregation) by eliminating the $2b$ values that are farthest from the CM. Zeno/Zeno++ [20], [21] are related algorithms that require an oracle at the server, which can generate an estimate of the true gradient in each iteration for screening/aggregation purposes. In particular, the screening procedure involves calculating a score for each local gradient based on its difference from the oracle gradient, with the surviving gradients being the ones that are the most similar to the oracle gradient.

The Byzantine-resilient distributed SGD framework in Algorithm 1 has also been investigated in [22] under a generalized Byzantine threat model. The basic assumption underlying this threat model is that the set of nodes under Byzantine attacks can change in each iteration of the algorithm, but the attackers can inject only malicious information into some dimensions of their respective gradients. Similar to [15] and [16], the work in [22] puts forth the use of CM (which is a variant of CTM) and geometric median for screening purposes, and establishes the resilience of the resulting methods under the assumption that less than half of the gradients in each dimension are attacked during each iteration.

We conclude by discussing a few methods that somewhat deviate from the distributed framework in Algorithm 1. In contrast to [15]–[22], the Byzantine–Robust Stochastic Aggregation (RSA) algorithm of [23] robustifies distributed SGD by making individual nodes store and update local versions of the global optimization variable $\mathbf{w}$, which are then aggregated at the server in each iteration in a Byzantine-resilient manner. To reduce the communications overhead of Byzantine-resilient distributed SGD, [24] discusses the signSGD algorithm, which uses only the element-wise sign of the gradient (rather than the gradient itself) for the update step. In particular, it is established in [24] that signSGD, with its element-wise majority vote on the signs of the local gradients as a screening/aggregation rule, is a Byzantine-resilient learning method. Finally, [25] proposes and analyzes a variant of distributed SGD that requires only one pass over the entire training data. Unlike other works, however, resilience in this article is accomplished through the explicit labeling of nodes as Byzantine, which are then excluded from all future computations at the server.

We now summarize key aspects of some of the Byzantine-resilient distributed learning algorithms in Table 2 in terms of the number of nodes M, the number of Byzantine nodes b,

Screening and Aggregation in Two Dimensions

We illustrate the robustness of different screening/aggregation methods against Byzantine attacks through a simple 2D example. Consider a total of seven 2D points (g_1, g_2), out of which six represent correct data (the black discs in Figure S3) and one has been falsified by a Byzantine node (the red disk in Figure S3). It can be seen from Figure S3, which is displayed with nonuniform g_1- and g_2-axes to capture the effect of the Byzantine node, that the ordinary average operation is highly susceptible to the falsified data point. Conversely, screening and aggregation rules such as coordinate-wise median, coordinate-wise trimmed mean, Krum, and Bulyan (using Krum in Figure S3) all produce final results that stay close to the true average of the faultless data points.

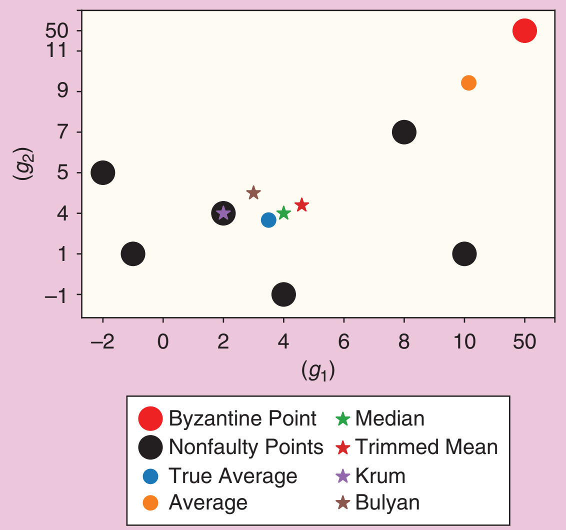

FIGURE S3. The effects of data falsified by Byzantine nodes on screening methods.

and the number of independent and identically distributed (i.i.d.) samples per node N. The convergence rates in the table correspond to the gap between the learned model after t iterations and the minimizer of the statistical risk [compare (2)] in the limit of large N. The parameter c denotes a constant that may change from one algorithm to the other, the $O(\cdot)$ indicates scaling that hides its dependence on problem parameters (including dimension d in the top part of the table), *N/A* signifies that an algorithm lacks a particular result, and — means that guarantees for an algorithm are not directly comparable to other algorithms. The last column in the top part of the table lists the conditions on M and b necessary for the well posedness of different algorithms. The bottom part of the table lists the per-iteration computational complexity of the screening procedure for algorithms that involve an explicit screening step at the server. Although the convergence/learning rates in the table are for strongly convex and smooth loss functions, some of the works require further assumptions and/or also provide results under a relaxed set of assumptions on the loss function and training data. We refer the reader to the works cited in this section for further details.

Numerical experiments

It can be seen from Table 2 that CM, CTM, and GeoMed have some of the best theoretical guarantees for strongly convex and smooth loss functions in terms of convergence and learning rates. We now numerically compare the performance of most of the algorithms listed in Table 2 on nonconvex loss functions. Our comparison excludes GeoMed because it lacks an algorithm for computing the exact geometric median of a set of high-dimensional gradients, as well as signSGD and RSA, because they differ from the rest of the algorithms in terms of their approach to Byzantine resilience.

The nonconvex learning task in the experiments corresponds to the classification of the 10 classes in the CIFAR-10 data set using a convolutional neural network with three convolution layers followed by two fully connected layers. Each of the three convolution layers are followed by rectified linear unit activation and max pooling, while the output layer uses softmax activation. The distributed setup corresponds to one server and $M = 20$ nodes, with the 50,000 sample training set uniformly at random distributed across the system (i.e., each node has $N = 2$ and 500 training samples). We run two rounds of experiments for each algorithm, where each round is repeated 10 times, with each trial corresponding to 20,000 algorithmic iterations, and the results on the CIFAR-10 test set are averaged over these 10 trials. In the first round, none of the nodes are taken to be Byzantine nodes. In the second round, four nodes are randomly selected as Byzantine nodes, with each sending a random vector to the server in each iteration whose elements uniformly take values in the range $(0, 10^{-5})$ for odd-numbered iterations and (0, 20) for even-numbered iterations. We limit ourselves to four Byzantine nodes to provide a fairer comparison between different algorithms, because five Byzantine nodes exceeds the theoretical limit of some algorithms (e.g., Bulyan). In both rounds of experiments, the algorithms operate under the assumption of $b = 4$.

We conclude our discussion of the experimental setup by noting that the optimal Byzantine attack strategy that adversely affects all distributed learning algorithms in a uniform manner remains an open problem. Nonetheless, the attack strategy being employed in our experiments has been carefully designed in light of the discussions in prior works (see, e.g., [12]); in particular, it appears to be the uniformly most potent strategy for the distributed learning algorithms under consideration in this article.

The results of the experiments in terms of average classification accuracy are shown in Figure 2(a) and (b), which highlight some tradeoffs that should help practitioners select the algorithms that best fit their needs. We first note from Table 2 that the computational complexity of screening steps in Krum and Bulyan scales quadratically with the number of nodes M, whereas it scales only linearly with M in median, trimmed mean, and Zeno/Zeno++. Next, it can be seen from Figure 2(a) that all of the screening-based methods fall short of the distributed SGD's performance in the faultless setting. This is in line with the conventional wisdom

Table 2. A summary of some recent results concerning Byzantine-resilient distributed machine learning.

Algorithm	Convergence Rate	Statistical Learning Rate	Condition on (*M*, *b*)
CM [15]	$O(c^t)$	$O\left(\frac{b}{M\sqrt{N}}+\frac{1}{\sqrt{MN}}+\frac{1}{N}\right)$	$M \geq 2b+1$
CTM [15]	$O(c^t)$	$O\left(\frac{b}{M\sqrt{N}}+\frac{1}{\sqrt{MN}}\right)$	$M \geq 2b+1$
GeoMed [16]	$O(c^t)$	$O\left(\frac{\sqrt{b}}{\sqrt{MN}}\right)$	$M \geq 2b+1$
Krum [17]	N/A	N/A	$M \geq 2b+3$
Multi-Krum [17]	N/A	N/A	$M \geq 2b+m+2$
Bulyan [18]	N/A	N/A	$M \geq 4b+3$
Zeno/Zeno++ [20], [21]	$O(c^t)+O(1)$	N/A	$M \geq b+1$
RSA [23]	$O\left(\frac{1}{t}\right)+O(1)$	N/A	$M \geq b+1$
signSGD [24]	—	N/A	$M \geq 2b+1$

Algorithm	CM, CTM, and Zeno/Zeno++	GeoMed	Krum and Multi-Krum	Bulyan
Screening complexity	$O(Md)$	$O\left(Md+bd\log^3\left(\frac{1}{\gamma}\right)\right)^*$	$O(M^2d)$	$O(M^2d+Md)$

*Screening computational complexity for GeoMed is for computing (1 + γ)-approximate geometric median [16].
N/A: not applicable.

that states that robustness often comes at the expense of correctness, with median and Krum paying the highest price in terms of correctness, while Bulyan, trimmed mean, and Zeno pay the least (and somewhat insignificant) price.

In the presence of Byzantine nodes, on the other hand, the vanilla distributed SGD completely falls apart. In contrast, while varying degrees of reduction in the performance of screening-based methods are observed under Byzantine attacks, none of them breaks down to the level of distributed SGD. Nonetheless, median, trimmed mean, and Krum (in this particular order) are affected the most in terms of performance by the aggressive Byzantine attack strategy employed in our experiments. Conversely, Bulyan and Zeno seem to be the most stable under Byzantine attacks. In addition, both of these methods offer a competitive tradeoff between correctness (in a faultless setting) and robustness (under attack). However, it is worth noting that Zeno's resilience comes at the expense of an oracle that can provide it with some knowledge of the true gradient. Similarly, because Bulyan screens four times the number of Byzantine nodes in each iteration, its resilience comes at the expense of the limited number of Byzantine nodes it can handle (compare Table 2).

Adversary-resilient decentralized processing of data

Distributed systems in general and distributed master–worker architectures in particular have become the workhorse frameworks for noncentralized processing of data. The reasons for this range from their relative ease of implementation and subsequent scaling up or scaling down of the system through the addition or removal of nodes to their relative simplicity of synchronization and communication protocols. Despite these and other related advantages, decentralized setups for inference and learning are increasingly being investigated in lieu of distributed setups for a multitude of reasons. Unlike the distributed setup, a decentralized system lacks a central server that is connected to every node [see Figure 1(b)]. Instead, all of the nodes in a decentralized system maintain local copies of the decision variable/machine-learning model and reach (an approximate) consensus on a common solution through the periodic exchange of messages over a network with a subset of other nodes, termed *neighbors*. Because of this and, unlike distributed systems, there is no single point of failure in decentralized systems. In the same vein, whereas the star communications topology in distributed master–worker architectures can create a communications bottleneck at the central server [see Figure 1(a)], the network/communications topologies in decentralized systems can be carefully designed to avoid such bottlenecks.

One of the biggest reasons for the study of Byzantine-resilient inference and learning in decentralized systems, however, is the emergence of inherently decentralized applications, such as the networks of self-driving cars and robot swarms. Typically, both the centralized and distributed setups cannot be engineered in a cost-effective manner in such applications, which are generally studied under the moniker of *multiagent systems*. This necessitates the use of decentralized frameworks, often with ad hoc network topologies, for statistical inference and machine learning.

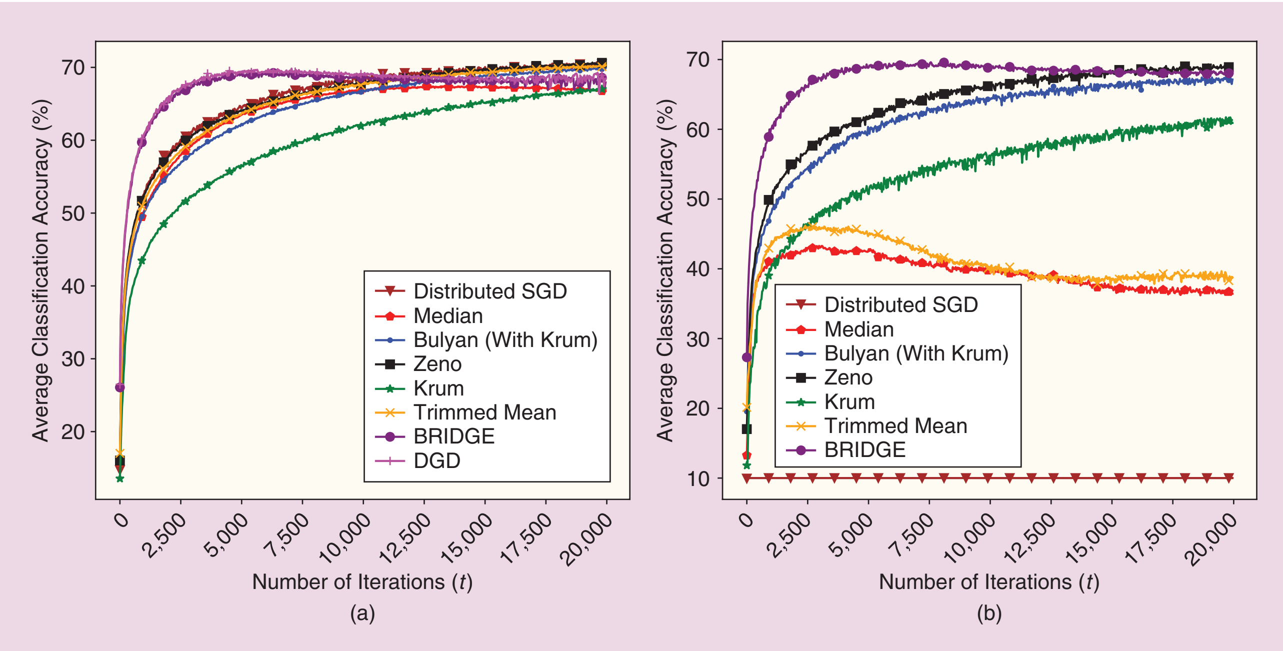

FIGURE 2. A comparison of different distributed learning methods (with median and trimmed mean being coordinate wise) based on SGD: (a) corresponds to the faultless setting, while (b) corresponds to the case of four Byzantine nodes in the system. In both cases, the parameter b is set equal to 4 in all Byzantine-resilient algorithms. We have also overlayed plots of two decentralized machine-learning methods, namely, BRIDGE and DGD, on top of the ones for distributed learning methods. These two methods are discussed in the "Decentralized Machine Learning" section. BRIDGE: Byzantine-resilient decentralized gradient descent; DGD: decentralized gradient descent.

Because no single node in decentralized setups gains access to the entire set of local variables, ensuring robustness against Byzantine failures is generally harder in decentralized systems. In particular, unlike simple characterizations of the feasible and/or necessary relationships between the number of nodes M and the number of Byzantine nodes b in Tables 1 and 2, the necessary and/or sufficient conditions for Byzantine-resilient decentralized algorithms are stated in terms of the topology of the underlying network graph. In addition, although the practical benefits of asynchronous update rules for and the impact of time-varying network topologies on decentralized processing of data have long been investigated by researchers, the literature on Byzantine-resilient decentralized processing is relatively sparse in this regard. Because of this and due to space constraints, our discussion in this article is mostly limited to synchronous algorithms on decentralized networks whose topology does not change with time (i.e., static graphs). In the following section, we begin by engaging in a discussion of an algorithmic process, aptly termed *consensus*, which often forms the basis of algorithms for decentralized inference and machine learning.

Decentralized consensus

Many decentralized inference and learning algorithms require a subprocess that ensures (approximate) agreement, i.e., consensus, among all the nodes. In the context of Byzantine-resilient decentralized processing, therefore, it is instructive to understand the fundamentals of Byzantine-resilient consensus. In this regard, our discussion will be limited to linear strategies for consensus, which form the basis for Byzantine-resilient decision consensus, averaging consensus, convex consensus, and so on [26], [27].

Suppose each node j in the decentralized system has a local variable $\mathbf{w}_j^0$ and it is desired to reach consensus on the average of these variables at each node through in-network message passing, i.e., $\mathbf{w}_j \approx (1/M)\Sigma_{i=1}^{M}\mathbf{w}_i^0$ for every node j. Consensus algorithms usually proceed by having each node iteratively take a weighted average of their neighbors' variables. Mathematically, therefore, a consensus algorithm is associated with an $M \times M$ weight matrix whose (j, i)th entry corresponds to the weight assigned by node j to the variable it receives from node i. In the absence of Byzantine nodes, a well-known result says that a doubly stochastic weight matrix whose smallest nonzero entry is lower bounded guarantees the convergence of all $\mathbf{w}_j$'s to the average $(1/M)\Sigma_{i=1}^{M}\mathbf{w}_i^0$. Suppose, however, that node k in the system is Byzantine and that it transmits $\mathbf{w}_k = \mathbf{w}'$ to its neighbors in each iteration. This simple "lazy" Byzantine strategy is enough for all the nodes to converge to $\mathbf{w}'$. And, if there were another Byzantine node k' in the system that had always transmitted the same $\mathbf{w}_{k'} \neq \mathbf{w}'$, the nodes in the system will not reach consensus at all.

The main idea behind ensuring the robustness of (averaging) consensus to Byzantine nodes is to screen potential outliers at each node before the weighting step in each iteration. Because the averaging of finite-dimensional vectors is equivalent to the averaging of their individual respective (scalar) coordinates, much of the discussion related to Byzantine-resilient consensus has focused on scalar-valued problems. Some variant of the trimmed mean, in which a node removes the largest b and the smallest b values received from its neighbors (including itself) is, in particular, a common screening method used in scalar consensus. Clearly, each node has to have enough neighbors (e.g., more than $2b$ for trimmed mean) for the screening-based consensus to be well defined; however, because the network must not become disconnected after screening, the exact constraint on network topology for different algorithms tends to be more involved (see [26] and [27] for further discussion of different screening methods and their corresponding topology constraints). Note that the decentralized inference and learning algorithms utilizing similar screening ideas for Byzantine resilience tend to have similar topology constraints.

In terms of the performance of Byzantine-resilient consensus, even the best screening/aggregation method cannot guarantee the removal of all falsified data. However, equipped with a reasonable strategy, a Byzantine-resilient scalar consensus algorithm can guarantee two things in each iteration: 1) the retained values are between the smallest and the largest scalars at all nonfaulty nodes; and 2) the difference between the largest and the smallest nonfaulty values decreases. Together, these two conditions can ensure that the nonfaulty nodes converge to a common value. However, because a node can also retain falsified data and/or eliminate nonfaulty data, consensus to the true average is no longer possible; instead, guarantees can be given only to an approximate average, such as a convex combination of initial values. This, in particular, is what makes Byzantine-resilient decentralized inference and learning so challenging.

We conclude by noting that asynchronous variants of decentralized consensus in the presence of Byzantine attacks have been investigated in [28] and [29]. The asynchronicity in [28] comes through the use of randomized gossip algorithms but, the focus in that work is primarily on the detection of Byzantine nodes. In contrast, [28] extends the synchronous (screening) framework and analysis of [26] to asynchronous settings (and time-varying network topologies). A noteworthy implication of [28] is that asynchronicity does not impose additional topology constraints required for resilience of decentralized consensus algorithms. This is important because synchronous algorithms often incur additional latency; indeed, a synchronous algorithm can only be as fast as the slowest connection/node in the system.

Decentralized statistical inference

Similar to the case of distributed inference, here we limit ourselves to decentralized detection and estimation. In both problems, decentralized consensus makes an integral part of the developed algorithms.

Decentralized detection

Akin to distributed detection, a typical setup in decentralized detection involves nodes in the system making observations under one of two (or more) hypotheses. Unlike distributed detection, however, nodes must collaborate among themselves to reach consensus in favor of one of the hypotheses. The vulnerability of the vanilla consensus framework to Byzantine nodes, therefore, also makes decentralized detection highly susceptible to Byzantine attacks.

Even though several application scenarios call for Byzantine-resilient decentralized detection, it has received less attention compared to its distributed counterpart. Some of the most relevant works in this regard include [30]–[32], all of which focus on scalar-valued problems. In the spirit of Byzantine-resilient scalar consensus, an adaptive threshold-based screening method is utilized in [30] to mitigate the impact of Byzantine nodes on the final decision. Similarly, a robust variant of distributed alternating direction method of multipliers (ADMM) is proposed in [31] and uses trimmed mean to eliminate 2 b scalars at each node in every iteration. But, other than the trivial topology constraint imposed by the trimmed mean (i.e., in-degree of nodes being greater than $2b$), this work lacks guarantees. In contrast, [32] proposes and analyzes a robust detection scheme that involves the identification of Byzantine nodes and the use of a weighted average consensus algorithm. However, this work focuses on a particular variant of Byzantine attacks and does not characterize topology constraints as a function of the number of Byzantine nodes.

Decentralized estimation

Decentralized estimation, in which nodes use local observations and messages from neighbors to reach consensus on the estimate of an unknown parameter $\mathbf{w} \in \mathbb{R}^d$, is often also studied under the linear model. Similar to decentralized detection, Byzantine-resilient decentralized estimation is a relatively recent topic of interest. Among the relevant works, [33] focuses on a specialized estimation problem in which node j need only estimate the jth entry of $\mathbf{w}$, and there are three types of nodes in the system: reliable, which are special nodes having perfect knowledge of the entry of $\mathbf{w}$ associated with them, ordinary (normal), and Byzantine. Under this setup and the assumption of a time-varying directed graph, [33] proposes and analyzes a trimmed mean-type (scalar-valued) screening procedure for Byzantine resilience. The works [34], [35] study vector-valued dynamic estimation under the noiseless observation model of $\mathbf{y}_j[n] = \mathbf{w}, n = 1, \ldots,$ for nonfaulty nodes. Although the threat model in [34] and [35] is a specialized variant of the Byzantine model, it allows for the indices of the nodes under attack to change from one time instance to the next. Both [34] and [35] use similar consensus+innovations algorithms coupled with time-varying gains to achieve resilience. The main difference in these works is that [35] can achieve linear convergence as opposed to sublinear convergence for [34], but it can tolerate only less than 30% of the nodes being under attack, in contrast to 50% for [34]. Finally, [36] accomplishes Byzantine-resilient decentralized dynamic estimation under the general linear model $\mathbf{y}_j[n] = \mathbf{H}_j \mathbf{w} + \boldsymbol{\eta}_j[n]$ by focusing on the explicit detection of adversaries. We conclude by noting that the review article in [2] provides more a detailed overview of [33], [34], and [36].

Decentralized machine learning

Decentralized machine-learning algorithms, which can be considered a combination of consensus and distributed learning frameworks, approximately solve (2) by minimizing a global loss function on the noncolocated data in a decentralized manner while reaching an agreement among all nodes, i.e.,

$$\min_{\{\mathbf{w}_1, \ldots, \mathbf{w}_M\}} \frac{1}{M} \sum_{j=1}^{M} f(\mathbf{w}_j, \mathbf{Z}_j) \quad \text{subject to} \quad \mathbf{w}_i = \mathbf{w}_j \; \forall i, j, \tag{3}$$

where $\mathbf{w}_j \in \mathbb{R}^d$ denotes the model learned at node j in the system. We refer to (3) as the *decentralized* ERM problem, which approximately solves the statistical risk minimization problem (2).

Here we focus on synchronous gradient descent-based methods for solving (3). The classic decentralized gradient descent (DGD) method [37], for instance, involves each node j exchanging its current local iterate in every iteration t with all the nodes in its neighborhood $\mathcal{N}_j$ and then updating the local iterate using a consensus-type weighted averaging step and a local gradient descent step, i.e.,

$$\mathbf{w}_j^{t+1} = \underbrace{\alpha_{jj} \mathbf{w}_j^t + \sum_{i \in \mathcal{N}_j} \alpha_{ji} \mathbf{w}_i^t}_{\text{consensus}} \underbrace{- \rho(t) \mathbf{g}_j(\mathbf{w}_j^t)}_{\text{local gradient descent}}, \tag{4}$$

where $\{\alpha_{ji}\}$ is the collection of averaging weights, $\rho(t)$ denotes the step size, and $\mathbf{g}_j(\mathbf{w}) := \nabla_{\mathbf{w}} f(\mathbf{w}, \mathbf{Z}_j)$ denotes the local gradient. Similar to the case discussed in the "Decentralized Consensus" section, however, a simple lazy Byzantine strategy at one or more nodes will lead to the breakdown of DGD-based learning methods.

Scalar-valued problems

The robustification of decentralized ERM-based learning in the presence of Byzantine attacks requires resilient variants of DGD. The works in [4] and [38] focus on this for scalar-valued (i.e., $d = 1$) decentralized optimization. Similar to distributed learning and decentralized consensus, resilience in these methods is also achieved through the use of a screening-based aggregation procedure within the consensus step in (4). The works in [4] and [38], in particular, focus on trimmed-mean screening: after node j receives $w_i \in \mathbb{R}$ from its neighbors, it removes the largest b and the smallest b w_is and takes average of the remaining scalars for consensus. This leads to the following update rule:

$$w_j^{t+1} = \frac{1}{\left| \mathcal{N}_{j^*}^t \right|} \sum_{i \in \mathcal{N}_{j^*}^t} w_i^t - \rho(t) g_j(w_j^t), \tag{5}$$

where $\mathcal{N}_{j^*}^t$ is the set of nodes (possibly including node j itself) that survive the screening at node j. This update can be shown to be Byzantine resilient in the sense that the w_j^ts converge to the minimizer of some convex combination of local losses $f(w, \mathbf{Z}_j)$. We also have a result from [4], suggesting that the minimum of the decentralized ERM in (3), even when restricted to the set of nonfaulty nodes, cannot be achieved in the presence of Byzantine nodes. Despite this negative result, it can be shown that any convex combination of local losses will converge in probability to the global statistical risk in the case of i.i.d. training samples. This can then be leveraged to establish the robustness of (5) for scalar-valued decentralized learning [39].

Vector-valued problems

The algorithms in [4] and [38] cannot be directly utilized in vector-valued problems (i.e., $\mathbf{w} \in \mathbb{R}^d$ for $d > 1$). On the one hand, unless a problem decouples over different coordinates of the optimization variable $\mathbf{w}$, minimizing the objective function along one coordinate independent of the other coordinates does not yield the right solution. On the other hand, because the trimmed-mean procedure of [4] and [38] requires the sorting of values received from one's neighbors, it cannot be directly applied to members of an unordered space like $\mathbb{R}^d$. This limitation of [4] and [38] is overcome in [39], which proposes an algorithm termed *Byzantine-resilient decentralized coordinate descent* (*ByRDiE*) for vector-valued decentralized learning in the presence of Byzantine nodes. ByRDiE, fundamentally a coordinate descent method, cyclically updates one coordinate at a time in a decentralized manner. And, because each subproblem in coordinate descent becomes a scalar-valued problem, ByRDiE uses trimmed-mean screening in each inner (coordinate-wise) iteration for Byzantine resilience. The final update rule in each coordinate-wise iteration of ByRDiE takes a form similar to (5), except that the gradient term also depends on other coordinates of $\mathbf{w}$.

For strictly convex and smooth loss functions, [39] guarantees (algorithmic and statistical) convergence of the iterates of ByRDiE to the statistical minimizer, with the algorithmic convergence rate being sublinear and the statistical learning rate being $O(a/\sqrt{MN})$; here, N again denotes the number of i.i.d. training samples per node, while the parameter a differs from one problem setup (and Byzantine attack model) to another and satisfies $1 \le a \le \sqrt{M}$. Despite these guarantees, which establish that ByRDiE can result in robust and (sample-wise) fast statistical learning in decentralized setups that exceed the local learning rate of $O(1/\sqrt{N})$ [39], the one-coordinate-at-a-time update of ByRDiE can be inefficient for high-dimensional (i.e., $d \gg 1$) problems. Indeed, because the coordinate-wise gradient update step depends on the updates of other coordinates, the iterates in ByRDiE cannot be updated in a coordinate-wise parallel fashion, leading to high network coordination and local computation overheads in decentralized learning.

To curtail the high overheads of ByRDiE, [40] presents another algorithm, termed *Byzantine-resilient decentralized gradient descent* (*BRIDGE*), which is based on gradient descent and CTM. Similar to DGD, each node j in BRIDGE also exchanges its entire current iterate $\mathbf{w}_j^t$ in every iteration t with all nodes in its neighborhood $\mathcal{N}_j$. The update step in BRIDGE, however, involves coordinate-wise screening/aggregation of $\mathbf{w}_j^t$s using trimmed mean with parameter $2b$, which is followed by a local gradient descent step. Mathematically, the (parallel) update of the kth coordinate is given by

$$\forall k \in \{1, \ldots, d\} \text{ (in parallel)},$$
$$w_j^{t+1}(k) = \frac{1}{\left|\mathcal{N}_{j^*}^{t,k}\right|} \sum_{i \in \mathcal{N}_{j^*}^{t,k}} w_i^t(k) - \rho(t) g_j^k(\mathbf{w}_j^t), \tag{6}$$

where $\mathcal{N}_{j^*}^{t,k}$ is the set of nodes whose kth coordinates survive the screening at node j and $g_j^k(\mathbf{w})$ denotes the kth coordinate of the local gradient $\mathbf{g}_j(\mathbf{w})$. In the case of strongly convex and smooth loss functions, theoretical guarantees for BRIDGE match those for ByRDiE.

Topology constraints

We noted previously (see the "Decentralized Consensus" section) that Byzantine resilience in decentralized setups depends on network topology. In this section, we describe two related topology constraints for trimmed mean-based decentralized learning. The constraint in [4] requires that, after removing all b Byzantine nodes and any combination of the remaining b (incoming) edges from each node, there is always a group of nodes, termed *source component*, of cardinality of at least $(b+1)$ that has a directed path to every other node. In other words, this constraint means that every nonfaulty node, even after trimmed-mean screening, can always receive information, directly or indirectly, from the source component. The topology constraints for ByRDiE and BRIDGE are also based on this condition. The constraint in [38], on the other hand, is based on the idea that any two arbitrary partitions of the network must result in one of the partitions having at least one node with $(2b+1)$ neighbors outside the partition. This, in turn, guarantees that b Byzantine nodes cannot isolate any subset of nonfaulty nodes during trimmed-mean screening.

Beyond trimmed-mean screening

Unlike distributed learning, where several screening methods and aggregation rules have been proposed and analyzed for Byzantine resilience (Table 2), the utilization of screening methods in decentralized learning has been limited to CTM. The main reason for this is the need for iterate consensus in decentralized learning, which makes the analysis of other screening/aggregation methods challenging. (It is also worth noting that our focus for this article is on iterate screening, as opposed to gradient screening in distributed learning.) In practice, it is possible to merge the ideas behind BRIDGE and screening methods such as CM, Krum, and Bulyan to develop other Byzantine-resilient decentralized learning methods. This is the approach we have taken in the numerical experiments discussed in the following section. Consensus, convergence analysis, and the statistical learning rates of such methods, however, remain an open problem.

Numerical experiments

We first compare the performance of DGD, ByRDiE, and BRIDGE and three variants of BRIDGE that come about from the incorporation of the screening principles of CM, Krum, and Bulyan (with Krum) within the BRIDGE framework. The computational inefficiency of ByRDiE for high-dimensional data sets and machine-learning models necessitates an experimental setup, both in terms of the data set and the learning task, that differs from the one utilized as part of our discussion on Byzantine-resilient distributed machine learning. Specifically, note that each (colored) image in the CIFAR-10 data set is 3,072-dimensional, and the corresponding convolutional neural network used earlier in our discussion for this data set gives rise to a model with $d = 122,410$. In contrast, in this article, we resort to a computationally tractable experimental setup that corresponds to the multiclass classification of the Modified National Institute of Standards and Technology (MNIST) database data set (784-dimensional data samples) using a linear one-layer neural network (i.e., $d = 7,840$). In addition to being computationally manageable for ByRDiE, this setup also results in a loss function that satisfies the theoretical conditions for the convergence of ByRDiE and BRIDGE.

The decentralized system in our experimental setup involves a total of $M = 20$ nodes in the network, with a communications link (edge) between two nodes decided by a random coin flip. Once a random topology is generated, we ensure that each node has at least $4b + 1$ nodes in its neighborhood (a condition imposed due to Bulyan screening). The training data at each node correspond to $N = 2,000$ samples randomly selected from the MNIST data set. The performance of each method is reported in terms of classification accuracy and averaged over $(M - b)$ nodes and a total of 10 independent Monte Carlo trials as a function of the number of scalars broadcast per node. The final results, shown in Figure 3, correspond to two sets of experiments: 1) the faultless setting in which none of the nodes actually behaves maliciously; and 2) the setting in which two of the 20 nodes are Byzantine, with each Byzantine node broadcasting every coordinate of the iterate as a uniform random variable between −1 and 0. Note that this Byzantine attack strategy is by no means the most potent in all decentralized settings; however, similar to the attack strategy utilized in our earlier discussion on distributed learning, this particular strategy has been selected after careful evaluation of the impact of different strategies proposed in works such as [12]–[14] on our particular experimental setup. Finally, with the exception of DGD, all of the methods are initialized with parameter $b = 2$ in both faultless and faulty scenarios.

It can be seen from Figure 3 that, other than ByRDiE and Krum-based screening, all of the methods perform almost as well as DGD in the faultless case. In the presence of Byzantine nodes, however, DGD completely falls apart, whereas the performances of all the screening methods remain comparable to the faultless setting. It is also worth comparing these results to those for Byzantine-resilient distributed learning (see Figure 2). Although CM and trimmed mean appear to be the worst performers in Figure 2, Krum-based screening is the least effective in Figure 3. In both cases,

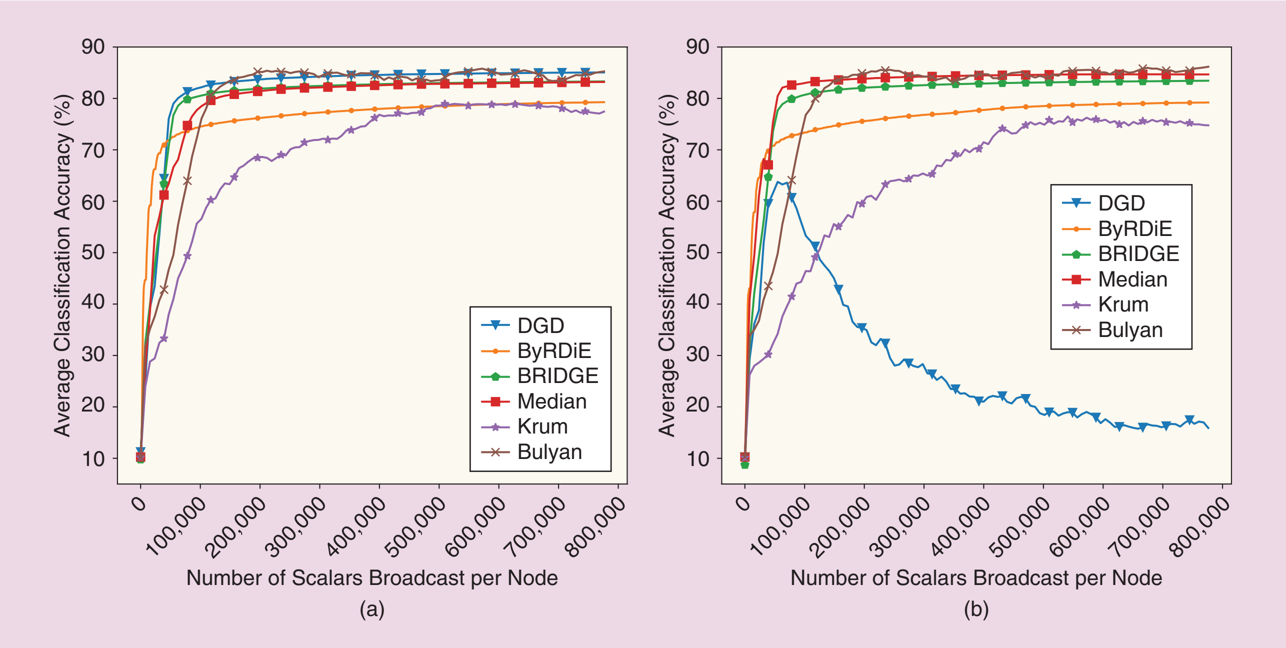

FIGURE 3. A performance comparison of different decentralized learning methods in (a) faultless and (b) Byzantine settings. Byzantine-resilient algorithms in both settings operate under the assumption of $b = 2$. Algorithms median, Krum, and Bulyan are effectively BRIDGE, combined with the screening procedures advocated in distributed learning.

however, Bulyan is quite effective, except that it has stringent topology requirements.

We conclude by explicitly comparing the performance of DGD and BRIDGE in decentralized settings to that of (Byzantine-resilient) learning methods in distributed settings. Because DGD and BRIDGE are scalable to high-dimensional learning tasks, our experimental setup, data set, data distribution, learning task, and Byzantine attack strategy for this comparison are identical to the ones described previously for Byzantine-resilient distributed learning methods. To ensure that the decentralized setup in the case of BRIDGE satisfies the topology constraints corresponding to the four Byzantine nodes in the system, we use 0.7 as the probability of random connectivity between any two nodes. The final results for DGD and BRIDGE, which are overlayed on top of the ones for distributed learning methods in Figure 2, show that BRIDGE offers competitive performance in both faultless and faulty systems. In fact, BRIDGE (and DGD) have faster convergence rates than the distributed learning methods for this particular nonconvex problem, decentralized setup, and Byzantine strategy. Although this could be partly attributable to the higher network connectivity of 0.7 in the decentralized setting, a careful head-to-head comparison and understanding of distributed and decentralized learning methods in both faultless and faulty settings is an open problem.

Some open research problems

Despite recent advances, Byzantine-resilient inference and learning remains an active area of research with several open problems. Much of the focus in distributed inference has been on a somewhat restrictive model in which Byzantine nodes do not collude. Collaborative Byzantine attacks, on the other hand, can be much more potent than independent ones. A fundamental understanding of mechanisms for safeguarding against such attacks remains a relatively open problem in distributed inference. Byzantine-resilient distributed estimation under nonlinear models is another problem that has been relatively unexplored. In the case of Byzantine-resilient distributed learning, existing works have only scratched the surface. The convergence and/or learning rates of many of the proposed methods remain unknown (Table 2). In addition, even though SGD is a workhorse of machine learning, approaches such as accelerated first-order methods (e.g., accelerated gradient descent), first-order dual methods (e.g., ADMM), and second-order methods (e.g., Newton's method) do play important roles in machine learning. However, the resilience of the distributed variants of such methods to Byzantine attacks has not been investigated in the literature.

The lack of a central server, the need for consensus, and an ad hoc topology make it even more challenging to develop and analyze Byzantine-resilient methods for decentralized inference and learning. Much of the work in this regard is based on screening methods such as trimmed mean and median that originated in the literature on Byzantine-resilient scalar-valued consensus. This has left open the question of how other screening methods, such as the ones explored within distributed learning, might handle Byzantine attacks, both in theory and in practice, in various decentralized problems. Unlike distributed learning, any such efforts will also have to characterize the interplay between network topology and the effectiveness of the screening procedure. The fundamental tradeoffs between the robustness and the (faultless) performance of Byzantine-resilient methods also remain largely unknown for decentralized setups. Finally, existing works on decentralized learning only guarantee sublinear convergence for strictly/strongly convex and smooth functions. Whether this can be improved upon by taking advantage of faster distributed optimization frameworks or different screening methods also remains an open question.

Conclusions

In this article, we presented an overview of the latest advances in Byzantine-resilient inference and learning. In the distributed master–worker setting, which is characterized by the presence of a central server that computes the final solution, we discussed recent results concerning the resilience of distributed detection, estimation, and machine learning against Byzantine attacks. Within distributed machine learning, we focused on Byzantine-resilient variants of distributed SGD, whose performances were compared using numerical experiments. In the decentralized setting, which typically requires consensus due to lack of a central server, we first discussed the principles behind Byzantine-resilient consensus. This was followed by a discussion of the latest results on decentralized detection, estimation, and learning in the presence of Byzantine nodes. We also compared and contrasted different Byzantine-resilient decentralized learning methods using numerical experiments and discussed the similarities and differences between them and those of distributed learning methods. Byzantine-resilient inference and learning has a number of research challenges that remain unaddressed, some of which are also briefly discussed in this article.

Acknowledgments

We gratefully acknowledge the support of the U.S. National Science Foundation (grants CCF-1453073 and CCF-1907658), the U.S. Army Research Office (grant W911NF-17-1-0546), and the DARPA Lagrange Program (the U.S. Office of Naval Research/Space and Naval Warfare contract N660011824020).

Authors

Zhixiong Yang (zhixing.yang@rutgers.edu) received his B.S. degree from Beijing Jiaotong University, China, in 2011, his M.S. degree from Northeastern University, Boston, Massachusetts, in 2013, and his Ph.D. degree from Rutgers University, New Brunswick, New Jersey, in 2020, all in electrical engineering. He is currently a system engineer at Blue Danube Systems. His research interests include distributed

processing, machine learning, and massive multiple-input, multiple-output systems.

Arpita Gang (arpita.gang@rutgers.edu) received her B.Tech. degree from the National Institute of Technology, Silchar, India, and her M.Tech. degree from the Indraprastha Institute of Information Technology, Delhi, India. She is currently pursuing her Ph.D. degree in electrical engineering at Rutgers University, New Brunswick, New Jersey. Her research interests include distributed optimization and signal processing. She is a Student Member of the IEEE.

Waheed U. Bajwa (waheed.bajwa@rutgers.edu) has been an associate professor in the Department of Electrical and Computer Engineering and the Department of Statistics at Rutgers University, New Brunswick, New Jersey, since 2011. His research interests include statistical signal processing, high-dimensional statistics, and machine learning. He is a Senior Member of the IEEE.

References

[1] A. Vempaty, L. Tong, and P. Varshney, "Distributed inference with Byzantine data: State-of-the-art review on data falsification attacks," *IEEE Signal Process. Mag.*, vol. 30, no. 5, pp. 65–75, 2013. doi: 10.1109/MSP.2013.2262116.

[2] Y. Chen, S. Kar, and J. M. F. Moura, "The Internet of Things: Secure distributed inference," *IEEE Signal Process. Mag.*, vol. 35, no. 5, pp. 64–75, 2018. doi: 10.1109/MSP.2018.2842097.

[3] J. Zhang, R. S. Blum, and H. V. Poor, "Approaches to secure inference in the Internet of Things: Performance bounds, algorithms, and effective attacks on IoT sensor networks," *IEEE Signal Process. Mag.*, vol. 35, no. 5, pp. 50–63, 2018. doi: 10.1109/MSP.2018.2842261.

[4] L. Su and N. Vaidya, Fault-tolerant distributed optimization (Part IV): Constrained optimization with arbitrary directed networks. 2015. [Online]. Available: arXiv:1511.01821

[5] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Progr. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982. doi: 10.1145/357172.357176.

[6] V. S. S. Nadendla, Y. S. Han, and P. K. Varshney, "Distributed inference with M-ary quantized data in the presence of Byzantine attacks," *IEEE Trans. Signal Process.*, vol. 62, no. 10, pp. 2681–2695, 2014. doi: 10.1109/TSP.2014.2314072.

[7] B. Kailkhura, Y. S. Han, S. Brahma, and P. K. Varshney, "Distributed Bayesian detection in the presence of Byzantine data," *IEEE Trans. Signal Process.*, vol. 63, no. 19, pp. 5250–5263, 2015. doi: 10.1109/TSP.2015.2450191.

[8] W. Hashlamoun, S. Brahma, and P. K. Varshney, "Audit bit based distributed Bayesian detection in the presence of Byzantines," *IEEE Trans. Signal Inform. Proc. Netw.*, vol. 4, no. 4, pp. 643–655, 2018. doi: 10.1109/TSIPN.2018.2806842.

[9] B. Kailkhura, S. Brahma, B. Dulek, Y. S. Han, and P. K. Varshney, "Distributed detection in tree networks: Byzantines and mitigation techniques," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 7, pp. 1499–1512, 2015. doi: 10.1109/TIFS.2015.2415757.

[10] J. Haupt, W. U. Bajwa, M. Rabbat, and R. Nowak, "Compressed sensing for networked data," *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 92–101, 2008. doi: 10.1109/MSP.2007.914732.

[11] A. Vempaty, O. Ozdemir, K. Agrawal, H. Chen, and P. K. Varshney, "Localization in wireless sensor networks: Byzantines and mitigation techniques," *IEEE Trans. Signal Process.*, vol. 61, no. 6, pp. 1495–1508, 2013. doi: 10.1109/TSP.2012.2236325.

[12] C. Xie, S. Koyejo, and I. Gupta, Fall of empires: Breaking Byzantine-tolerant SGD by inner product manipulation. 2019. [Online]. Available: arXiv:1903.03936

[13] M. Fang, X. Cao, J. Jia, and N. Z. Gong, Local model poisoning attacks to Byzantine-robust federated learning. 2019. [Online]. Available: arXiv:1911.11815

[14] G. Baruch, M. Baruch, and Y. Goldberg, "A little is enough: Circumventing defenses for distributed learning," in *Proc. Advances in Neural Inf. Process. Syst. (NeurIPS'19)*, 2019, pp. 8632–8642.

[15] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. 35th Int. Conf. Machine Learning (ICML'18)*, 2018, vol. 80, pp. 5650–5659.

[16] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," in *Proc. ACM Measurement and Analysis of Computing Systems*, Dec. 2017, vol. 1, pp. 44:1–44:25. doi: 10.1145/3154503.

[17] P. Blanchard, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. Advances in Neural Inf. Process. Syst. (NeurIPS'17)*, 2017, pp. 118–128.

[18] E. El-Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in Byzantium," in *Proc. 35th Int. Conf. Machine Learning (ICML'18)*, 2018, vol. 80, pp. 3521–3530.

[19] E. El-Mhamdi and R. Guerraoui, Fast and secure distributed learning in high dimension. 2019. [Online]. Available: arXiv:1905.04374

[20] C. Xie, O. Koyejo, and I. Gupta, Zeno: Byzantine-suspicious stochastic gradient descent. 2018. [Online]. Available: arXiv:1805.10032

[21] C. Xie, O. Koyejo, and I. Gupta, Zeno++: Robust asynchronous SGD with arbitrary number of Byzantine workers. 2019. [Online]. Available: arXiv:1903.07020

[22] C. Xie, O. Koyejo, and I. Gupta, Generalized Byzantine-tolerant SGD. 2018. [Online]. Available: arXiv:1802.10116

[23] L. Li, W. Xu, T. Chen, G. Giannakis, and Q. Ling, "RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets," in *Proc. AAAI Conf. Artificial Intelligence (AAAI'19)*, 2019, vol. 33, pp. 1544–1551. doi: 10.1609/aaai.v33i01.33011544.

[24] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, signSGD with majority vote is communication efficient and Byzantine fault tolerant. 2018. [Online]. Available: arXiv:1810.05291

[25] D. Alistarh, Z. Allen-Zhu, and J. Li, "Byzantine stochastic gradient descent," in *Proc. Advances in Neural Inf. Process. Syst. (NeurIPS'18)*, 2018, pp. 4618–4628.

[26] H. J. LeBlanc, H. Zhang, X. Koutsoukos, and S. Sundaram, "Resilient asymptotic consensus in robust networks," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 4, pp. 766–781, 2013. doi: 10.1109/JSAC.2013.130413.

[27] N. Vaidya, "Matrix representation of iterative approximate Byzantine consensus in directed graphs. 2012. [Online]. Available: arXiv:1203.1888

[28] L. Haseltalab and M. Akar, "Approximate Byzantine consensus in faulty asynchronous networks," in *Proc. IEEE American Control Conf. (ACC'15)*, 2015, pp. 1591–1596. doi: 10.1109/ACC.2015.7170960.

[29] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Stochastic and deterministic fault detection for randomized gossip algorithms," *Automatica*, vol. 78, pp. 46–60, Apr. 2017. doi: 10.1016/j.automatica.2016.12.011.

[30] S. Liu, H. Zhu, S. Li, C. Chen, and X. Guan, "An adaptive deviation-tolerant secure scheme for distributed cooperative spectrum sensing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM'12)*, 2012, pp. 603–608. doi: 10.1109/GLOCOM.2012.6503179.

[31] B. Kailkhura, P. Ray, D. Rajan, A. Yen, P. Barnes, and R. Goldhahn, "Byzantine-resilient locally optimum detection using collaborative autonomous networks," in *Proc. IEEE 7th Int. Workshop Computational Advances Multi-Sensor Adaptive Process. (CAMSAP'17)*, 2017, pp. 1–5. doi: 10.1109/CAMSAP.2017.8313059.

[32] B. Kailkhura, S. Brahma, and P. K. Varshney, "Data falsification attacks on consensus-based detection systems," *IEEE Trans. Signal Inform. Proc. Netw.*, vol. 3, no. 1, pp. 145–158, 2017. doi: 10.1109/TSIPN.2016.2607119.

[33] H. J. LeBlanc and F. Hassan, "Resilient distributed parameter estimation in heterogeneous time-varying networks," in *Proc. 3rd Int. Conf. High Confidence Networked Systems (HiCoNS'14)*, 2014, pp. 19–28. doi: 10.1145/2566468.2566476.

[34] Y. Chen, S. Kar, and J. M. F. Moura, "Resilient distributed estimation: Sensor attacks," *IEEE Trans. Autom. Control*, vol. 64, no. 9, pp. 3772–3779, 2018. doi: 10.1109/TAC.2018.2882168.

[35] Y. Chen, S. Kar, and J. M. F. Moura, "Resilient distributed estimation: Exponential convergence under sensor attacks," in *Proc. IEEE Conf. Decision Control (CDC'18)*, 2018, pp. 7275–7282. doi: 10.1109/CDC.2018.8619746.

[36] Y. Chen, S. Kar, and J. M. F. Moura, "Resilient distributed estimation through adversary detection," *IEEE Trans. Signal Process.*, vol. 66, no. 9, pp. 2455–2469, 2018. doi: 10.1109/TSP.2018.2813330.

[37] A. Nedić and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, 2009. doi: 10.1109/TAC.2008.2009515.

[38] S. Sundaram and B. Gharesifard, "Distributed optimization under adversarial nodes," *IEEE Trans. Autom. Control*, vol. 64, no. 3, pp. 1063–1076, 2019. doi: 10.1109/TAC.2018.2836919.

[39] Z. Yang and W. U. Bajwa, "ByRDiE: Byzantine-resilient distributed coordinate descent for decentralized learning," *IEEE Trans. Signal Inform. Proc. Netw.*, vol. 5, no. 4, pp. 611–627, 2019. doi: 10.1109/TSIPN.2019.2928176.

[40] Z. Yang and W. U. Bajwa, BRIDGE: Byzantine-resilient decentralized gradient descent. 2019. [Online]. Available: arXiv:1908.08098

SP