Bristle: Decentralized Federated Learning in Byzantine, Non-i.i.d. Environments

Joost Verbraeken

BRISTLE: Decentralized Federated Learning in Byzantine Non-i.i.d. Environments

Master thesis submitted to Delft University of Technology

in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in **Computer Science**

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)

by

Joost Verbraeken

Student number: 4475208

To be defended in public on 1st July, 2021

Graduation committee

First supervisor	: Dr. J.A. Pouwelse, Distributed Systems
Second supervisor	: Prof. Dr. M. Larson, Multimedia Group

An electronic version of this thesis is available at http://repository.tudelft.nl.

Contents

Abstract v

Preface vii

Main article 1

Introduction

System and Threat model

Bristle: Middleware for Decentralized Federated Learning
- Bootstrapping Bristle with Transfer Learning . . . .
- Distance-based Prioritizer . . . .
- Performance-based Integrator . . . .
- Implementation . . . .

Experimental Setup

Experimental Evaluation
- Performance Gains of Transfer Learning . . . .
- Byzantine Attacks . . . .
- Non-i.i.d. classes . . . .
- Combining Byzantine Attacks and Non-i.i.d. Classes . . . .
- Bristle Efficiency and Scalability . . . .

Related Work

Concluding Remarks

References

Supplementary Material 17

A Preliminaries to Federated Learning 19
- A.1 Basic idea behind federated learning 19

B Related work **21**
B.1 Byzantine-attacks 21
B.1.1 Data poisoning vs model poisoning attacks 21
B.1.2 Untargeted vs targeted attacks 22
B.2 Byzantine-resilience 23
B.2.1 Distance-based screening 24
B.2.2 Performance screening 26
B.2.3 Pruning 27
B.2.4 Behavioral-based 28
B.2.5 Other 29
B.3 Non-i.i.d. data 31
B.4 Communication-efficiency 33

C Implementation **35**
C.1 Environment 35
C.2 Network protocol 36
C.3 Experiment orchestration and deployment 36
C.4 Initiation of tests 37
C.4.1 User-interface 37
C.4.2 Command-line arguments 38
C.4.3 Network request 38
C.5 Optimizations 38

D Discussion **41**
D.1 The road ahead 41
D.2 Biggest issues encountered 44
References 46

Abstract

Federated learning (FL) is a type of machine learning where devices locally train a model on their private data. The devices iteratively communicate this model to a central server which combines the models and sends the updated model back to all devices. Because the data stays on the devices and only the model is transmitted, federated learning is considered as a privacy-friendly alternative to regular machine learning where all data is transmitted over the internet.

However, the central server used in typical FL systems not only poses a single point of failure susceptible to crashes or hacks, but may also become a performance bottleneck. These issues are alleviated by decentralized FL (DFL), where the peers communicate model updates with each other instead of with a single server.

Unfortunately, DFL is challenging since (1) the training data possessed by different peers is often non-i.i.d. (i.e., distributed differently between the peers) and (2) malicious, or Byzantine, attackers can share arbitrary model updates with other peers to subvert the training process.

We address these two challenges and present *Bristle*, middleware between the learning application and the decentralized network layer. Bristle leverages transfer learning to predetermine and freeze the non-output layers of a neural network, significantly speeding up model training and lowering communication costs. To securely update the output layer with model updates from other peers, we design a fast distance-based prioritizer and a novel performance-based integrator. The prioritizer prioritizes the model updates based on their distance to the peer's own model and an explore-exploit trade-off, and the integrator integrates each class of each model update separately based on their performance on a small set of i.i.d. test samples. Their combined effect results in high resilience to Byzantine attackers and the ability to handle non-i.i.d. classes.

We empirically show that Bristle converges to a consistent 95% accuracy in Byzantine environments, outperforming all evaluated baselines. In non-Byzantine environments, Bristle requires 83% fewer iterations to achieve 90% accuracy compared to state-of-the-art methods. We show that when the training classes are non-i.i.d., Bristle significantly outperforms the accuracy of the most Byzantine-resilient baselines by 2.3x while reducing communication costs by 90%.

PREFACE

A year ago, I thought I had the perfect idea for a master thesis. I wanted to work on a universal decentralized marketplace that would disrupt the way economies worked, and it would be built on a blockchain. And of course, at the Delft University of Technology it is pretty much impossible to say the word "blockchain" without saying the words "Johan Pouwelse". So, I presented my original idea, learned that I might have overlooked a few key aspects, changed the subject, changed the subject a bit more, changed the subject much more, actually changed the subject to something completely different in a few months time, and started studying the research field that would become the topic of my thesis: Byzantine-resilient federated learning. The scientific field is incredibly young, the state-of-the-art can clearly be improved in many different ways, and the use cases are endless - which is important if you want to write a captivating introduction.

The results of the thesis are interesting and truly an important step forward. The algorithm that I devised (called Bristle) is very unique. It converges so incredibly fast that I had to modify the baselines because otherwise the results would be incomparable with other solutions. In terms of communication efficiency, no baseline comes even close to Bristle when considering the number of models transmitted, not to mention the total number of bits transmitted. When the classes are non-i.i.d. (not evenly distributed between the peers), Bristle blows the other baselines out of the water.

Of course, there are some limitations, the most notable of which are the dependence on a proper dataset that is suitable for transfer learning, and the inability to handle situations where the data within a single class is non-i.i.d. between the peers. However, I truly believe that Bristle can inspire other researchers with its innovative and effective architecture and serve as an important step towards the ultimate decentralized federated learning system.

In hindsight, I'm quite happy with the entire thesis process. Sometimes it was intense because I combined working on my thesis with a role as project manager / director at a consultancy organization, and sometimes I spent weeks fixing terrible bugs, the effort of which would not be mentioned anywhere in the resulting paper. However, I could always rely on the excellent feedback and support of Dr. Johan Pouwelse who helped to guide me throughout the entire master-thesis process. A special thanks to Dr. Martijn de Vos who not only gave superb feedback on a regular basis, but also helped a lot to polish the final paper and frame it in such a way that it would appeal specifically to the reviewers of the Middleware conference. I would also like to thank Prof. Martha Larson for her excellent feedback on my paper.

Main article

Bristle: Decentralized Federated Learning in Byzantine, Non-i.i.d. Environments

Anonymous Author(s)

Abstract

Federated learning (FL) is a privacy-friendly type of machine learning where devices locally train a model on their private data and typically communicate model updates with a server. In decentralized FL (DFL), peers communicate model updates with each other instead. However, DFL is challenging since (1) the training data possessed by different peers is often non-i.i.d. (i.e., distributed differently between the peers) and (2) malicious, or Byzantine, attackers can share arbitrary model updates with other peers to subvert the training process.

We address these two challenges and present *Bristle*, middleware between the learning application and the decentralized network layer. Bristle leverages transfer learning to predetermine and freeze the non-output layers of a neural network, significantly speeding up model training and lowering communication costs. To securely update the output layer with model updates from other peers, we design a fast distance-based prioritizer and a novel performance-based integrator. Their combined effect results in high resilience to Byzantine attackers and the ability to handle non-i.i.d. classes.

We empirically show that Bristle converges to a consistent 95% accuracy in Byzantine environments, outperforming all evaluated baselines. In non-Byzantine environments, Bristle requires 83% fewer iterations to achieve 90% accuracy compared to state-of-the-art methods. We show that when the training classes are non-i.i.d., Bristle significantly outperforms the accuracy of the most Byzantine-resilient baselines by 2.3x while reducing communication costs by 90%.

CCS Concepts: • **Computing methodologies** → *Lifelong machine learning*; *Transfer learning*; *Supervised learning by classification*; *Neural networks.*

Keywords: Decentralized Federated learning, Deep Transfer Learning, Gradient Aggregation Rule, Byzantine-resilience

ACM Reference Format:
Anonymous Author(s). 2021. Bristle: Decentralized Federated Learning in Byzantine, Non-i.i.d. Environments. In *22st International Middleware Conference (Middleware '21), December 6–10, 2021, Québec, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/1122445.1122456

Middleware '21, December 6–10, 2021, Québec, Canada
2021. ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 Introduction

Machine learning applications have gained significant traction and are widely used for various purposes, such as understanding consumer preferences [10], recognizing pictures [42], predicting keystrokes [14], or translating texts [17]. Traditionally, these applications use neural networks trained on a single server using a tremendous amount of data generated by a large number of geo-distributed, heterogeneous edge devices such as smartphones, IoT devices, or autonomous vehicles [35]. However, centralized training on data streams generated by such devices is limited by the following three factors. First, transmitting training data over the Internet can pose a significant burden on backbone networks. This burden is particularly problematic when media such as photos or videos are used as training data and is worsened by the fact that most learning applications communicate with cloud providers over wireless links [46, 78]. Second, maintaining a central server architecture can quickly become expensive and time-consuming as the number of devices increases [24]. Third, transmitting personal and sensitive information over the Internet, such as text conversations or photos, to cloud providers raises privacy concerns and is in certain jurisdictions not even allowed by regulations such as the US HIPAA [28] and the European GDPR law [75].

Federated Learning (FL) overcomes these three limitations. With FL, edge devices do not send their data to the server training the model but instead communicate model updates to a so-called *parameter server*, also see the left side of Figure 1 [15]. The parameter server coordinates the learning process by pushing model updates to devices (step ①), and peers train the model with their private data on-device (step ②). Then they send the updated model back to the parameter server (step ③), after which the server *aggregates* the incoming model updates into a global model (step ④). FL sidesteps the need for the data to leave the device, improving privacy, lowering the computational burden for the server, and decreasing the communication overhead when the model update is smaller than the data to be transmitted per iteration. FL is nowadays used for various applications, including next-word prediction on keyboards [2, 11, 13, 32], speech recognition [64], wireless communications [12, 59], human activity recognition [66], and health applications [9, 54, 61, 63].

The centralized architecture shown in Figure 1 is typical for FL systems. However, even though the parameter server synchronizes the learning process between remote edge devices, this approach comes with significant drawbacks [7, 51, 84]. Notably, the parameter server not only

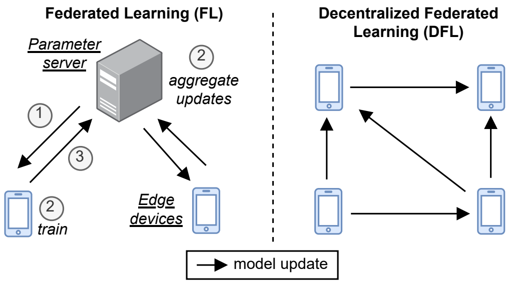


Figure 1. Federated learning using a parameter server (left) and decentralized federated learning (right).

poses a single point of failure susceptible to crashes or hacks, but it may also become a performance bottleneck as the number of devices pushing model updates increases [51]. These issues motivate further research to remove dependency on the parameter server and train models in a decentralized manner [22, 44, 51, 73]. With *decentralized federated learning* (DFL), each peer is connected to a subset of the other peers in the network from which it receives incoming models and to which it pushes its updated models [58]. We visualize this approach on the right side of Figure 1.

DFL has resulted in a new wave of learning methods that achieve comparable model accuracy as state-of-the-art FL approaches [34] while boasting several significant advantages. First, decentralized networks are fault-tolerant by design since peers continuously update their knowledge about which other peers are online or stopped communicating[76]. Second, decentralized networks are self-scaling. When a new peer joins the network, other peers can start communicating with the newly-joined peer seamlessly. Because there is usually a maximum number of other peers with which each peer communicates, the connection ratio will drop, but the network will typically still be strongly connected [27]. Third, DFL unlocks AI with zero-cost infrastructure (from the developer's perspective) since the computational power needed to train the network is delivered by all peers in the network working together instead of a parameter server under centralized control [25].

Limitations of DFL. Despite the benefits of DFL, we identify three challenges that reduce its potential for practical applications. The first barrier is that both existing FL and DFL architectures have to consider *Byzantine attacks*, the situation where malicious peers aim to undermine the model's training by purposefully sending specific model updates to the parameter server or other peers [90]. Since malicious peers keep their data private, they can easily "poison" their model and disseminate this malicious model without repercussions. Byzantine attacks are often addressed by the Gradient Aggregation Rule (GAR), which aggregates and combines incoming model updates [6, 56, 90]. At the same time, state-of-the-art GARs typically assume that the majority of peers act honestly, which is hard to guarantee in networks with open participation [88].

The second challenge is that the performance of conventional FL systems degrades significantly when being deployed in an environment where the distribution of the training data differs between peers. This is also known as *non-i.i.d.* (not independent and identically distributed) data. Given that data is typically non-i.i.d. in an FL environment [6, 14, 90], dealing with non-i.i.d. data is considered a *key challenge* in FL [38]. The inability of most GARs to handle non-i.i.d. data results in Catastrophic Forgetting, a phenomenon where the peers overwrite model parameters that were important to predict a particular class distribution with parameters essential to predict another class distribution [91].

The third challenge relates to the communication requirements to disseminate model updates among peers in the network. In DFL, each peer disseminates a copy of the current model at every training iteration to several other peers. However, modern neural networks may consist of millions of parameters [33, 36], sometimes requiring gigabytes of data to be transferred for each model exchange. Facilitating this amount of data quickly becomes infeasible when the number of devices and the frequency of model updates increase [62].

Although some DFL systems are to a certain extent Byzantine-resilient [6, 31, 56, 85], able to deal with non-i.i.d. data [15, 65, 93], or focus on reducing the communication costs [3, 21, 80], there exists - to the best of the author's knowledge - no DFL solution that addresses all three challenges simultaneously. We argue that such a solution is a key requirement for a wide deployment of DFL systems.

Our Contributions. To the best of the author's knowledge [31, 85, 86, 94], only four papers have ever been published about DFL that are Byzantine-resilient (first challenge), none of which are suitable for situations where the data is non-i.i.d. (second challenge) or decrease the communication requirements (third challenge). Therefore, we present *Bristle*: the first Byzantine-resilient and communication-efficient approach for DFL in environments with non-i.i.d. classes.[1] Bristle is a pure edge solution and acts as a middleware between the machine learning application and the decentralized network layer by combining three techniques:

1. Using *deep transfer learning*, model training with Bristle converges quickly and achieves high accuracy with low communication overhead and while requiring low amounts of on-device data (Section 3.1).
2. With a fast *distance-based prioritizing step* during model aggregation based on an explore-exploit trade-off, Bristle pre-filters incoming model updates that are estimated to improve the current model (Section 3.2).

[1]`Bristle` is an acronym for "Byzantine-Resilient mIddleware for StochasTic federated LEarning".

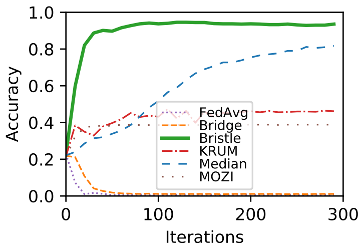


Figure 2. Our Bristle middleware for decentralized federated learning outperforms state-of-the-art solutions in a Byzantine and non-i.d.d. environment.

3. With a novel *performance-based integrator*, Bristle provides state-of-the-art Byzantine-resilience, even when the classes are unevenly spread across peers (non-i.i.d.). Unlike related solutions, this component performs per-class performance measurements instead of per-model measurements (Section 3.3).

Similar to related work in the FL domain, Bristle focuses on supervised learning problems that train a neural network for classification [55, 56, 81]. We implement all elements of Bristle as an Android library and open-source our implementation (see Section 3.4).

With an extensive set of experiments with the popular MNIST dataset, we evaluate the performance and resilience of Bristle. We compare Bristle with five related approaches for FL and quantify the Byzantine-resilience of these approaches under four attacks in the domain of FL, aimed at reducing the model's accuracy. The experiments show that even in highly Byzantine environments where the classes are non-i.i.d., Bristle not only withstands all evaluated attacks but also outperforms all related approaches in terms of convergence speed, accuracy, consistency, and communication efficiency. We highlight one of the key results from our experiments in Figure 2, showing model accuracy as peers train the model. In this figure, the performance of Bristle and evaluated DFL approaches is illustrated in a Byzantine (40% of the peers perform a label-flip attack) and non-i.i.d. environment (the class overlap between the peers is just 40%). Despite this challenging environment, Bristle quickly converges and outperforms all other approaches in terms of accuracy while reducing communication costs by over 90%.

2 System and Threat model

We now describe our system and threat model, and state the assumptions underlying our work.

Network Model. We consider an unstructured, strongly connected peer-to-peer network with n peers. Each peer knows the network addresses of other peers in the network. We assume unreliable and unordered network channels between peers. In addition, we do not place any restriction on peers joining the system, and peers can join or leave the system at any time. We consider the mitigation of attacks targeted at the network layer, e.g., the Eclipse Attack, beyond the scope of this work.

Training Data. Each participating peer i acts on local training dataset D_i which size is denoted by $|D_i|$. Training data never leaves the device, and D_i is only known to peer i. In contrast to most related work, we assume that the number of samples per class is not distributed uniformly among peers (non-i.i.d), which is a key characteristic of FL environments [6, 14, 90]. However, we also assume that for each class the samples are distributed uniformly (i.i.d.) between peers as a prerequisite for the performance-based integrator (see Section 3.3).

Model Training. Our approach focuses on supervised classification problems where the dataset consists of a collection of input-output pairs. The input is a chunk of data with a fixed size, and the output is a qualitative label associated with a class. The devices aim to minimize the loss function by tuning the parameters in the neural network such that its predictive capabilities are maximized. We use the negative log-likelihood of the ground truth class as the loss function. Since this problem is intractable for complex models, we use, in line with most literature [47, 70, 71, 90], a technique called Gradient Descent (GD) which iteratively takes the derivative of the loss function with respect to the training data and then moves the hyperparameters in the direction of the gradient. However, because the local dataset can be large, it can take a long time for GD to converge [8]. Therefore, we use the faster Stochastic Gradient Descent (SGD), where a subset (mini-batch) of five samples is stochastically sampled from the dataset to update the parameters in a particular iteration.

Threat Model. Our work assumes an environment with *an unconstrained number of Byzantine attackers* aiming to subvert the model's performance. This includes the Sybil Attack, an attack where a malicious peer joins the network under many different identities to prevent model convergence [23]. It also includes collusion, the situation where malicious peers in concert attempt to undermine model convergence. Byzantine attackers can send arbitrary model updates to any peer in the network, but have no access to the private samples of benign peers.

3 Bristle: Middleware for Decentralized Federated Learning

Bristle enables DFL that can handle non-i.i.d. classes and thwart Byzantine attacks while improving communication costs compared to existing approaches. We provide a high-level overview of Bristle and the actions performed during a training iteration in Figure 3. First, Bristle takes advantage

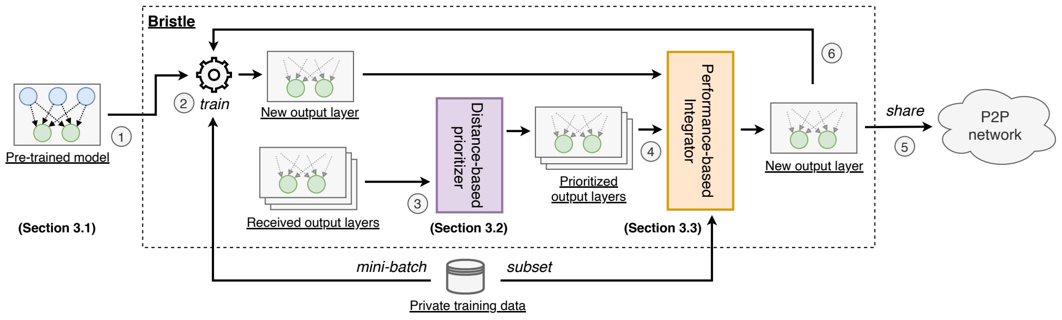


Figure 3. High-level overview of Bristle, our middleware for decentralized federated learning.

of deep transfer learning to determine the non-output layers of each node before the first training iteration (step ①). As we will experimentally demonstrate in Section 5, this significantly reduces communication costs compared to communicating the entire model, speeds up the model convergence rate, and acts as a prerequisite to learning on non-i.i.d. classes.

The main innovation of Bristle is its two-phased GAR that integrates received models. During each training iteration, a peer i trains the local model on a mini-batch of their private dataset D_i and only updates the output layers (step ②). It then forwards the updated output layer[2] and the output layers it has received from other peers to a distance-based prioritizer (step ③). This fast distance-based prioritizer estimates the best candidates for integration based on an explore-exploit ratio and forwards them to the performance-based integrator (step ④). The performance-based integration is more computationally demanding and integrates the prioritized output layers into the peer's current output layer. This integration process is both Byzantine-resilient and capable of continual learning, facilitated by selective updating of the output layer, per-class performance evaluations, and a carefully crafted weighted averaging function. Our main motivation to use a two-staged approach is that a distance-based prioritizer on itself is ineffective and performance-based integration on all received models is too computationally expensive since it requires the evaluation of incoming models on private data. Finally, Bristle transmits the output layer to a few connected peers (step ⑤) and uses the new output layer as input for the next training iteration (step ⑥). In the remainder of this section, we elaborate on the model pre-training, the distance-based prioritizer, and the performance-based integrator.

[2]We use the terms output layer and model interchangeably in this work.

3.1 Bootstrapping Bristle with Transfer Learning

In conventional FL settings, the parameters of the entire neural network are shared with the parameter server or, when using DFL, with other peers. These parameters include all weights and biases of the input layer, hidden layers, and the output layer. In Bristle, we avoid exchanging the full neural network between peers. Instead, we leverage a popular ML technique called *deep transfer learning* which re-uses a neural network that has been trained on another dataset with comparable low-level features as the training data (step ① in Figure 3) [72]. More specifically, we copy and subsequently freeze the non-output layers from another model. Bristle then only trains and exchanges the output layer, which has three major advantages:

1. Copying the non-output layers from a well-trained model significantly *improves the convergence rate* when training the model in a decentralized fashion on devices with lower hardware capabilities.
2. Freezing non-output layers *reduces communication overhead* since only the output layers have to be shared among peers.
3. Freezing the non-output layers is a key step towards *continual learning*, which further increases the performance and robustness of Bristle (see Section 3.3).

To bootstrap Bristle with a pre-trained model, we must assume that for the dataset we want to train on, there exists a vastly bigger dataset with roughly the same low-level features that we can use to pre-train a similar model. Considering the extent to which transfer learning is nowadays used in practical learning problems, we argue that this assumption is realistic and does not prohibitively degrade the applicability of Bristle [1, 60]. The training of the initial neural network can be performed offline by system designers or volunteers. The pre-trained model can then be shared with peers by bundling them in the (mobile) application or be served by a trusted server that sends interested peers the

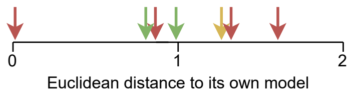


Figure 4. Average Euclidean distance from a given model to benign (green) and malicious (red) models, and models trained on an entirely different data distribution (orange).

pre-trained model upon request. We assume that the developer knows the number of classes beforehand to determine the size of the output layer.

3.2 Distance-based prioritizer (DBP)

We now elaborate on the distance-based prioritizer (DBP) in Bristle (step (2) in Figure 3). The inputs to the DBP are the output layers received from other peers during the last training iteration. To explain the motivation behind the DBP, we first examine Figure 4 where the average Euclidean distance is shown from a non-i.i.d. model (configured as explained in Section 5.3) to two benign models (shown in green), a benign model that is trained on an entirely different data distribution (shown in orange), and several Byzantine models (shown in red). The distances in this figure are based on the models exchanged during our experiments (see Section 5). For each distance shown in Figure 4, we took the average distance over 100 runs (with the current and the other model both trained ten times on different training data) and trained all models to convergence before comparing the distances. The figure shows that based on the Euclidean distance alone, we cannot reliably determine if a certain model is benign because, in this example, the distance to a Trimmed Mean attack model is between the distances to two benign models. However, the distance may clearly help prioritizing certain models since models with rather low or high distances appear to be more likely to be malicious.

Based on this insight, we segment the received models into three equally-sized categories with a low, medium, and high distance from our current model. Then, we uniformly sample models from each of these categories. The number of models

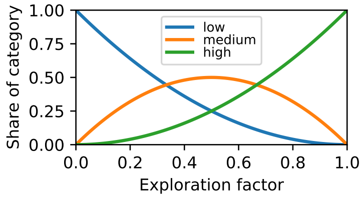


Figure 5. Proportion of models sampled by the distance-based prioritizer from each of the three categories, based on the exploration-exploitation ratio α.

sampled from each category depends on the exploration-exploitation ratio α where $\alpha = 0$ exclusively samples low-distance models (exploitation-dominant) and $\alpha = 1$ exclusively samples high-distance models (exploration-dominant). The optimal value of α depends on the maximum distance that benign models can reasonably have to each other, which is highly dependent on the degree of asynchrony (higher asynchrony $\rightarrow$ higher α), the expected ratio of benign to Byzantine peers (more Byzantine peers $\rightarrow$ lower α), and the degree of non-i.i.d.-ness (higher non-i.i.d.-ness $\rightarrow$ higher α). The advantage of using three categories is that in situations where the accuracy is clearly sub-optimal and a significant number of Byzantine attackers are present, the developer may want to emphasize integrating models with a medium distance since these will often perform best. f_l, f_m and f_h denote the fraction of samples from the low, medium and high distance categories, and are determined as follows:

$$f_l = (1-\alpha)^2, f_m = -2\alpha^2 + 2\alpha, f_h = \alpha^2 \quad (1)$$

Note that $f_l + f_m + f_h = 1$. Figure 5 visualizes the relation between the above values and α. We also parameterize the maximum number of models that pass our DBP, denoted by β. Since the runtime of our performance-based integrator scales with the input size, devices with lower performance can opt for a lower value for β.

3.3 Performance-based Integrator (PBI)

The ability of our distance-based prioritizer to recognize Byzantine or *stale* models that are trained on a lower number of iterations is rather limited. Byzantine attacks are easy to launch in open networks, and stale models are a regular phenomenon in FL systems [49]. To address these concerns, we assess the performance of each model that passes the distance-based prioritizer on a small *test dataset* (step (4) in Figure 3) using a performance-based integrator (PBI). This test dataset is a random subset of the full private dataset owned by the peer.

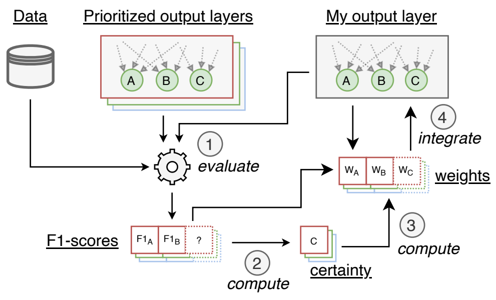


Figure 6. The per-class PBI in Bristle. Dashed boxes refer to values associated with foreign classes.

Figure 6 shows how the PBI works, given a model with three classes (A, B and C). In summary, the PBI selectively integrates the parameters of classes that perform well to achieve Byzantine-resilience and to handle non-i.i.d. classes properly. This is different from existing approaches that integrate each received model as a whole instead. We refer to the parameters in the output layer that are connected to a particular class as *Class-specific parameters (CSPs).* The integration process in Bristle first evaluates the performance of each class of each received model on a test dataset (step ①). Based on this evaluation, we compute a *certainty* score that estimates how benign a received model is (Step ②). Then, we calculate for all CSPs of each model a weight (step ③). Finally, we integrate the received models into our current model, based on the computed weights (step ④).

The selective integration of parameters is inspired by the continual learning algorithm CWR* [52] that enables non-i.i.d. learning by initializing the output layer to zero, applying a mean-shift, and replicating the hippocampus-cortex duality by selectively copying and resetting parts of the output layer. Related work demonstrates that this approach provides excellent performance in regular non-FL environments and enables per-class updates [52]. Applying continual learning requires that the non-output layers are frozen and equal between all peers. We already addressed this by using transfer learning (see Section 3.1) to train these layers before the first iteration.

We now elaborate on the PBI logic, which pseudocode is given in Listing 1. The input for this algorithm is the current model of the peer (*myModel*), and the prioritized models that passed the DBP (*prioritizedModels*).

Step 1 (per-class performance measurements). Based on a subset of the local dataset, we first calculate the F1-score for the CSPs of all *familiar classes* of both the peer's own output layer and the prioritized output layers (line 4-8). We refer to a class as familiar when the peer has a sufficient number of samples, denoted by κ, to reliably estimate the performance of that class of a given model. This threshold is determined by the developer. With a higher value of κ, the PBI can more reliably determine F1-scores, but the number of classes for which insufficient samples are available to evaluate, also known as *foreign classes*, might decrease. We chose to measure F1-scores instead of the accuracy as a proxy for the performance since the former is more suited for imbalanced datasets [16]. The data subset used by the PBI is never used to train the peer's own model since this would result in an overestimation of the performance of the peer's own model. Since these measurements depend on the availability of sufficient private data samples, Bristle avoids integrating models from other peers until sufficient private data samples are available to test them reliably.

Step 2 (certainty computation). Then, we calculate for each prioritized model a *certainty* score, which is a (rough) estimate of the degree to which this model is benign (line 10-12). This certainty score is determined by subtracting the standard deviation from the average F1-score of the ϕ best-performing familiar classes (line 11-12), thus rewarding high class performance and punishing high variation among the class performance. We do not consider all classes when computing the certainty score since it is not realistic to assume that all classes of each benign received model perform well in a non-i.i.d. environment. The larger ϕ is, the more robust the algorithm is against Byzantine attacks, but the less able it is to learn about foreign classes in the case of a non-i.i.d. setting.

Step 3 (weight computation). Furthermore, we estimate for each familiar class of each prioritized model if integrating its corresponding CSP improves the model's performance by simply checking if the F1-score of that class of the received model exceeds the respective F1-score of the peer's own model (line 14). We call the extent to which it does the *F1-discrepancy*, and we store this value in a two-dimensional array named *disc*. If the class-specific F1-score of the model being evaluated exceeds that of our own model, we calculated the F1-discrepancy as in line 15. Otherwise, we set the F1-discrepancy to $-\infty$ (line 17). The parameter η increases the degree to which high-performing classes are integrated.

The `computeFaWg` function then computes the weights for the familiar classes for each model, taking the prior computed scores and certainty as input (line 19). This function computes the following Sigmoid function, where w_c is the weight of familiar class c, s is the F1-discrepancy, and r is the certainty of the model being considered:

Algorithm 1 Bristle's performance-based integrator (PBI)

```
 1: procedure PBI(myModel, prioritizedModels)
 2:     F1 ← [], certainty ← [], disc ← []
 3:     faWg ← [], foWg ← []                          ▷ foreign/familiar class weights
 4:     for m in myModel ∪ prioritizedModels do                              ▷ Step 1
 5:         for c in familiarClasses(data) do
 6:             F1[m][c] ← evaluate(m, c, testData)
 7:         end for
 8:     end for
 9:
10:     for m in prioritizedModels do
11:         best ← sorted(F1[m]) [:$\phi$]
12:         certainty[m] ← max(avg(best) - std(best), 0)                     ▷ Step 2
13:         for c in familiarClasses(data) do                                ▷ Step 3
14:             if F1[m][c] $\geq$ F1[myModel][c] then
15:                 disc[m][c] ← (|F1[m][c] - F1[myModel][c]| * $\eta$)$^3$
16:             else
17:                 disc[m][c] ← $-\infty$
18:             end if
19:             faWg[m][c] ← computeFaWg(disc[m][c], certainty[m])
20:         end for
21:         foWg[m] ← computeFoWg(disc[m], certainty[m])
22:     end for
23:                                                                          ▷ Step 4
24:     myModel.integrateFamiliarClasses(prioritizedModels, faWg)
25:     myModel.integrateForeignClasses(prioritizedModels, foWg)
26: end procedure
```

$$w_c = max(0, \frac{\omega^1_{fa}}{1 + e^{-\frac{s}{100}}} - \omega^2_{fa}) * r \tag{2}$$

This function assigns a weight of one to equally performing classes and an increasingly higher weight to classes that show excellent performance, dependent on the values of ω^1_{fa} and ω^2_{fa}. ω_{fa} represents the boost for above-average performing models and the bounty for below-average performing models. The bigger this discrepancy, the faster the model can catch up with other better-performing models, but the bigger the impact of a malicious model that performs well on familiar classes and bad on foreign classes. We multiply the outcome of the Sigmoid function by the certainty score calculated earlier because the parameters of a class might be sub-optimal even when it has a perfect F1-score when other classes perform poorly. Figure 7 illustrates how the weight of CSPs is affected by their F1-score on a test dataset, assuming that the peer's current model yields a F1-score of 0.5 on that class and that the calculated certainty is 1.

Determining a weight for the CSPs of the foreign classes is challenging because we cannot directly measure their performance. Instead, we take the sum of the scores of all familiar classes and feed this into the same Sigmoid function as used for the familiar classes (Equation 2), albeit parameterized with separate variables ω^1_{fo} and ω^2_{fo}. This is done by the `computeFoWg` function (line 21). ω_{fo} is the extent to which foreign classes are integrated into the model. The higher this value, the better the model can be when the peer wants to use the model to classify formerly foreign classes, but also the higher the impact when a potentially malicious model is integrated. If the user is uninterested in achieving high performance on foreign classes, ω^1_{fo} can be set to 0.

Step 4 (model integration). Finally, we replace the CSPs of familiar and foreign classes with the weighted average of all CSPs by using the calculated weights (line 24-25).

3.4 Implementation

Bristle is implemented on top of an existing network library that provides support for peer discovery, decentralized overlay creation, and authenticated messaging [67, 68]. Since we envision the usage of Bristle mostly in a mobile environment, our middleware is written in the Kotlin programming language (the default language for Android applications). We

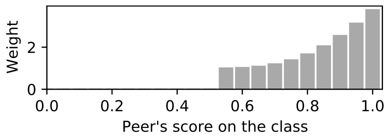


Figure 7. Weight assigned to the CSPs given an F1-score of 0.5 and a certainty score of 1.0.

envision that Bristle runs as a background service on the end-users' devices to periodically receive or transmit models from and to peers when the device is connected to Wi-Fi. Peers transmit model updates using the UDP protocol and network messages are compressed with Gzip. Model training is facilitated by the DeepLearning4j library (version 1.0.0-beta7) to enjoy compatibility with a wide range of advanced ML algorithms. We have published the Bristle source code and developer documentation in a GitHub repository.[3]

Using Bristle. Developers can leverage the Bristle middleware by feeding the peer's current model and all received models into Bristle's GAR after each iteration and subsequently replacing the peer's model with the result. The moment a model is updated can be decided by the developer, for example, model training can take place when the device is charging to minimize the impact on end users. The peers should be able to selectively receive and integrate only models relevant to the current ML application. To this end, Bristle uses the functionality to form communication groups as provided by its network library. Deciding on the variables listed under Section 4 is a key part of machine learning. We recommend the developer to use heuristics, optimize them on a related dataset, or use A/B-testing to find the best values to obtain the desired accuracy on the models.

4 Experimental Setup

We now describe our experimental setup, datasets, and parameters.

Testbed. All experiments are run on an HPE DL385 Gen10 server. This server is equipped with 128 AMD EPYC 7452 CPUs, has 512 GB of DDR4 memory, and runs Debian 10.

Datasets. In line with related research [55, 56, 81], we consider an image classification application that applies Convolutional Neural Networks (CNNs) to classify images. We use the popular MNIST [45] dataset, consisting of 60,000 gray-scale training images and 10,000 test images of 28x28 pixels representing handwritten digits. To achieve better performance, we standardize the dataset by applying Z-score normalization such that the features are re-scaled to a normal distribution with $\mu = 0$ and $\sigma = 1$. We pre-train - until convergence - a neural network with the same configuration on the EMNIST-Letters [18] dataset, which features resemble MNIST, but contains characters instead of digits. We then include the resulting neural network in the Bristle software.

Experiment parameters. We list all default parameters used during our experiments in Table 1. An exploration-exploitation ratio α of 0.4 slightly prioritizes model with lower distance.

Non-i.d.d. data. To evaluate Bristle with non-i.d.d. data, we first sort the data per class, divide the data into equally-sized shards, and then assign to every peer several shards,

[3]Source code can be requested through the Program Committee (to comply with the double-blind review requirements).

Experiment parameter	**Default value**
Environment	
Peers (n)	10
Connection ratio	100%
Fraction Byzantine peers	50%
Machine learning	
Mini-batch size (b)	5
Learning rate (λ)	0.001
L2-value	0.005
Max. iterations	300
Bristle	
Exploration-exploitation rate (α)	0.4
Max. PBI input size (β)	30
#familiar class selection size (ϕ)	3
#test samples per class (κ)	10
η	10
ω^1_{fa}, ω^1_{fo}	10
ω^2_{fa}, ω^2_{fo}	4

Table 1. Default parameters and values used during the experiments.

which is also the approach taken by related work [19, 77, 79]. We note that the more shards we assign to every peer, the better every peer can recognize and defend against Byzantine attacks (see Section 3.3), but the less non-i.i.d. the classes are. We opt to assign - unless specified otherwise - without loss of generality to every peer four shards which cover 40% of the classes to balance between the ability to learn non-i.i.d. classes and to recognize Byzantine attacks.

Model training. We train the dataset on the same CNN architecture used by McMahan et al. [55], except that we use Leaky ReLu instead of the regular ReLu as the activation function for the hidden layers since the former seems to give slightly better performance (specifically, suffers less from the vanishing gradients problem). The model consists of twice a convolutional layer (kernel size = 5, stride = 1, padding = 0) followed by a max pooling layer (kernel size = 2, stride = 2, padding = 0), and finally the output layer (with 800 hidden nodes). For the output function, we use the softmax function, and as the loss function, we use negative log-likelihood. To train the model, we use the Adam optimizer, parameterized as shown in Table 1. Since these values significantly impact the model's performance, we used a grid search with typically 7-9 values for each parameter and ensured that the optimal values were approximately in the middle. Although the baselines were originally not developed to be used in combination with transfer learning, we decided to use transfer learning for all baselines for all experiments anyway since the performance increase is so significant that otherwise any comparison with Bristle would be meaningless (also see Section 5.1).

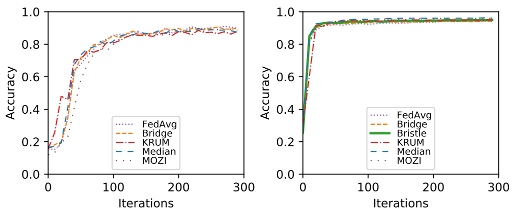
Baselines. To compare Bristle with existing methods, we implemented five other GARs commonly used in FL:

1. *FedAvg* [55] aggregates all models by coordinate-wise averaging of parameters. It is commonly used as a baseline to compare the performance of FL systems.
2. *Median* [90] aggregates all models by taking the coordinate-wise mean of parameters. As demonstrated in the literature, it is already a particularly effective Byzantine-resilient GAR [87].
3. *Krum* [6] integrates the model that most closely resembles (in terms of Euclidean distance) all other models as the new global model. Even if the selected model is malicious, in theory, the performance should not degrade too much as it is close to all other models.
4. *BRIDGE* [85] is specifically designed for Byzantine-resilient model aggregation in decentralized settings. It cyclically updates every coordinate one by one and subsequently applies trimmed-mean screening to obtain the final coordinate for each dimension.
5. *MOZI* [31] uses a combination of a fast distance-based and accurate performance-based filter to aggregate model updates in a Byzantine-resilient manner.

We initialized Krum and BRIDGE, which are dependent on a-priori knowledge of the number of attackers, with $b = 4$ (the maximum number of attackers) and Mozi with $\rho = 0.5$ (the ratio of benign to Byzantine peers).

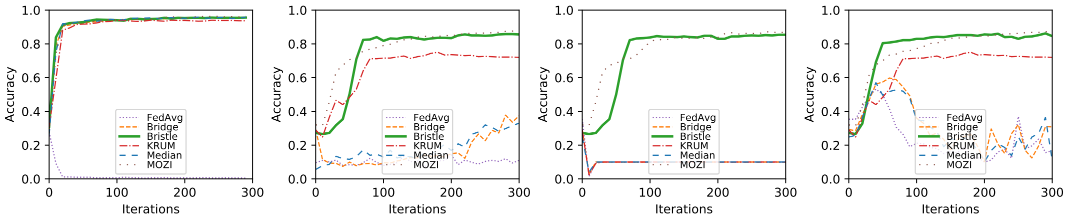
Byzantine attacks. We also evaluate Bristle under the following four Byzantine attacks, which are commonly considered in the domain.

1. The *Label-flip attack* [4] assigns an incorrect label to each input [4, 29, 74]. We implement this attack by numbering all labels and reassigning each sample with label x to label $(x + 1)$ % $|x|$.
2. The *Additive noise attack* [48] adds some noise to the parameters of outgoing models. When the noise has a larger variance, it can indeed prevent convergence but also makes the noise attack easier to detect [48]. Centering the noise around a value slightly different from 0 allows the attack to prevent convergence despite



(a) Regular learning **(b)** Transfer learning

Figure 8. The accuracy of the model while training, for regular and transfer learning.



(a) Label-flip attack **(b)** Additive noise attack **(c)** Krum attack **(d)** Trimmed Mean attack

Figure 9. The resilience of Bristle and other GARs against various Byzantine attacks, with 50% of all peers being malicious and i.i.d. data.

low variance, but since the mean of benign updates is always centered around 0, this attack can be easily detected. We consider a variant where each half of the parameters are set to noise centered around a value just below and above 0, respectively.

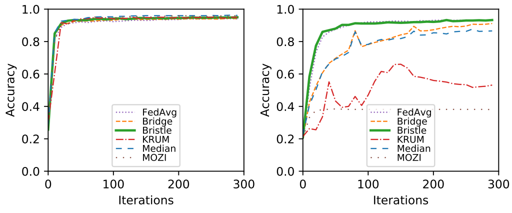
3. The *Krum attack* [26] specifically targets the Krum aggregation rule. It is an effective, state-of-the-art attack by iteratively sending attack vectors that will be accepted by Krum whilst inflicting maximum damage to the peer's model.
4. The *Trimmed Mean attack* [26] targets the trimmed mean GAR (Bridge in our experiments). It determines the gradient direction for each parameter of the model and then creates an attack vector that points exactly in the opposite direction, scaled per parameter depending on the values of the other benign peers.

5 Experimental Evaluation

We now evaluate the performance of Bristle. Our evaluation answers the following questions: *(1) what is the achieved training speedup when applying transfer learning to DFL? (2) How does Bristle perform in the presence of Byzantine attackers in terms of model accuracy? (3) How does Bristle perform when classes are not uniformly distributed over peers (non-i.i.d.) in terms of model accuracy? (4) How does Bristle perform in an environment with both Byzantine attackers and non-i.i.d. classes? And (5) What are the communication and computational costs of Bristle?*

5.1 Performance Gains of Transfer Learning

We first evaluate the performance gains of transfer learning when all peers are honest, and the data is i.i.d. For each experiment, we measure after every 10 iterations the performance of all peers and then take the average accuracy, defined as $\frac{\#correct\ predictions}{\#test\ samples}$. Figure 8 shows the evolution of model accuracy for the baseline GARs and Bristle, both without transfer learning (Figure 8a) and with transfer learning (Figure 8b). We note that Bristle requires a pre-trained model and therefore is not included in Figure 8a. Figure 8a shows that the evaluated GARs perform quite similarly, although Mozi has a slower convergence rate since it assigns lower weights to benign models that have not been trained sufficiently yet to perform well. Transfer learning dramatically improves both the convergence rate and the maximum accuracy after 300 iterations of all GARs, which also supports literature [37]. More specifically, whereas reaching 70% accuracy takes on average 55 iterations with regular training, it takes merely four iterations with transfer learning, a reduction of 93%. Reaching 90% accuracy takes on average 180 iterations with regular learning, whereas it only takes 30 iterations with transfer learning, a reduction of 83%. This reduction in iterations to reach the desired level of model accuracy can significantly reduce the load on the network.

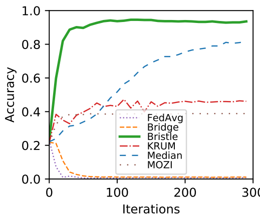


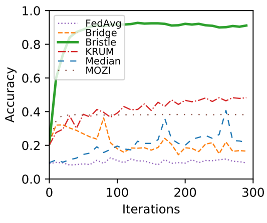
(a) All peers own samples of all classes **(b)** Each peer owns samples of 40% of the classes

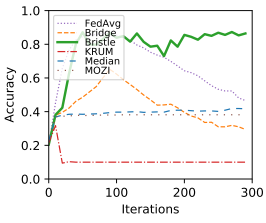
Figure 10. The performance of Bristle and other GARs when the data is i.i.d. and non-i.i.d.

5.2 Byzantine Attacks

We now evaluate the Byzantine-resilience of Bristle under different attacks when data is i.i.d. Figure 9 shows the effect of different attacks. Bristle withstands all evaluated Byzantine attacks and quickly achieves high model accuracy. Figure 9a shows that the label-flip attack prevents model convergence when using the FedAvg GAR, but is, despite its relatively small influence on the model's parameters, successfully mitigated by all other GARs. Figure 9b shows that the FedAvg


(a) Label-flip attack


(b) Additive noise attack


(c) Krum attack

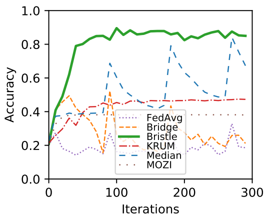
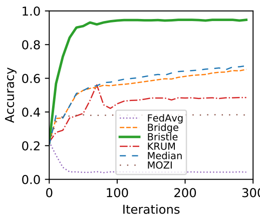
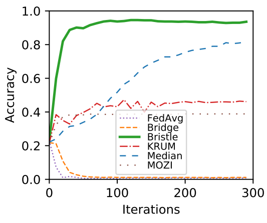
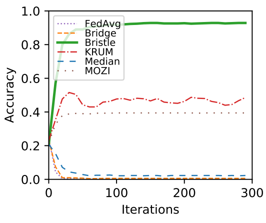

(d) Trimmed Mean attack

Figure 11. The resilience of Bristle and other GARs against various Byzantine attacks when the data is non-i.i.d.


(a) 10% of the peers is malicious


(b) 30% of the peers is malicious


(c) 50% of the peers is malicious

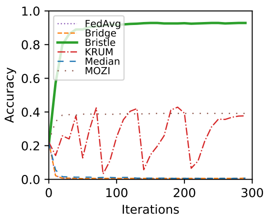

(d) 70% of the peers is malicious

Figure 12. The resilience of Bristle and other GARs under a label-flip-attack, with a varying percentage of all peers being malicious, where the data is non-i.i.d.

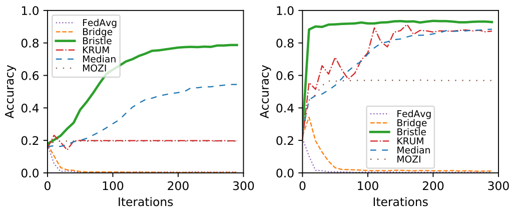
and Median GARs are susceptible to the additive noise attack. The Krum attack, shown in Figure 9c, is clearly very effective against Krum but has only a minor effect on the convergence rate of the other GARs. The Trimmed Mean attack, see Figure 9d, is relatively ineffective against any Byzantine-resilient GAR. This is because all benign models are very close to each other in this scenario, making it hard for this attack to steer the model in another direction without clearly being an outlier.

5.3 Non-i.i.d. Classes

We now consider the scenario where the classes are non-i.i.d., see Figure 10. In this experiment, there are 10 peers, each of which has access to two random samples of four consecutive classes. Thus, each peer i has access to classes $\{i, (i+1)\%10, (i+2)\%10, (i+3)\%10\}$. When we compare Figure 10b with Figure 10a, it is clear that Krum and Mozi fail to achieve desired model accuracy with non-i.i.d. classes, and we also observe that Bridge and Median converge significantly slower. FedAvg and Bristle show excellent performance and achieve 90% model accuracy in 65 and 55 iterations, respectively. This is because these methods (eventually) combine the information learned by every peer, while the other methods disregard a part of the received models under the incorrect assumption that those are Byzantine.

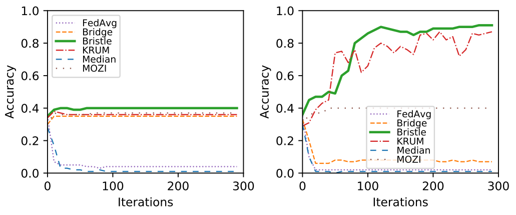
5.4 Combining Byzantine Attacks and Non-i.i.d. Classes

We now evaluate Bristle and other GARs in an environment containing both Byzantine attackers and where the classes are non-i.i.d. (see Figure 11). We use the label-flip attack in all experiments because this one is effective (see Figure 11a) and very popular in related work [26, 57, 82]. Except for Bristle, all baselines fare poorly in the Byzantine experiments with non-i.i.d. classes. Specifically, we observe that FedAvg is unable to defend against Byzantine attacks, although it achieves an accuracy of 87% in the first few iterations of the Krum attack (see Figure 11c). Although Bridge performs well in the experiments presented in Section 5.2, it performs poorly under the threat of Byzantine attacks when the classes are non-i.i.d. Krum is, as expected, unable to defend against the Krum attack but performs under the threat of the other attacks roughly on par with Mozi. Mozi shows the same accuracy as in the benign non-i.i.d. experiments (see Figure 9) because it successfully defends against Byzantine attacks but is also maxed out on the poor maximum accuracy that it can attain on the node's own dataset. The performance of the Median baseline varies significantly depending on the attack and may be quite inconsistent (as illustrated in Figure 11b and Figure 11d). An interesting finding is that the Trimmed Mean attack is relatively ineffective in the i.i.d. experiment (see Figure 9d) but impacts the performance of the same baselines



(a) Each peer has data of 20% of the classes **(b)** Each peer has data of 60% of the classes

Figure 13. The resilience of Bristle and other GARs under a label-flip-attack, with 30% of all peers being malicious, where the classes are to a varying extent non-i.i.d.



(a) Each peer is connected to 2% of the other peers **(b)** Each peer is connected to 5% of the other peers

Figure 14. The resilience of Bristle and other GARs under a label-flip-attack, with a varying percentage of the peers connected, where the classes are non-i.i.d.

significantly in the non-i.i.d. experiment (see Figure 11d). This results from the greater distance between the benign models in the non-i.i.d. experiment, giving the attack more leeway to steer the model in a different direction without being considered an outlier. Since the PBI assigns a weight of 0 to the malicious CSPs, Bristle consistently outperforms all other GARs and defends well against all evaluated Byzantine attacks.

Varying the Number of Byzantine Attackers. We now vary the number of Byzantine attackers, see Figure 12. Figure 12a shows that even with 10% Byzantine attackers, the performance of the baselines already degrades significantly. Bristle manages to maintain a quick increase in accuracy for all considered attack scenarios. With 10% and 30% Byzantine attackers, Mozi manages to keep a stable performance but is limited to predicting only the peer's familiar classes correctly. Figure 12d shows that the performance of Krum is inconsistent and that only Bristle achieves a consistent high performance. Bristle's excellent performance results from the fact that (a) in contrast to Mozi, Bristle evaluates and integrates the parameters per class instead of per model, and (b) in contrast to the other GARs, Bristle uses performance evaluations instead of just distance comparisons.

Varying the Degree of Non-i.i.d.-ness. We now compare the performance of the GARs in three situations where the classes are to a varying extent non-i.i.d. (see Figure 13). FedAvg and Bridge are, regardless of the degree of non-i.i.d.-ness, unable to provide any resilience to the label-flip attack. Mozi defends, similarly to the experiments in Figure 11, perfectly well against the label-flip attack but is limited to the maximum accuracy that can be obtained by training on its own dataset. The performance of Krum increases with the number of classes owned by the peer. The Median rule performs relatively well, even when the classes are highly non-i.i.d. Bristle, however, clearly performs best compared to all baselines, although its performance decreases when the peers have only 20% of the data (see Figure 13a).

Varying the Connection Ratio. In larger networks, it is unrealistic to assume all-to-all dissemination of model updates. To test the impact of the connection rate on the convergence rate more accurately, we set up an environment with 100 peers where the classes are again non-i.i.d. similarly to the previous experiments. We setup two experiments, connect each peer to a random subset of 2% and 5% of the other peers respectively, add an equal number of label-flip attackers that are connected to each benign peer, and measure the average accuracy over time. From Figure 14a, we observe that when the connection rate is only 2%, FedAvg and Median perform very poorly, but also the other GARs are unable to achieve satisfactory accuracy. When the connection rate increases to 5% in Figure 14b, Bristle and Krum perform quite well in contrast to the other baselines. Thus, it seems that in a setting with 100 peers, a connection rate of only 5% is already almost enough to reach the maximum accuracy. The number of iterations required to obtain this accuracy is relatively high compared to a connection rate of 100% (see Figure 11a).

5.5 Bristle Efficiency and Scalability

Figure 15 shows for each GAR the average time it takes for a peer to finish a single iteration depending on the number of connected peers. For evaluation purposes we assume that each peer receives from every other peer exactly one model

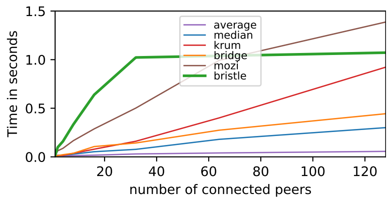


Figure 15. Average time to complete a single training iteration on the non-i.i.d. label-flip experiment with a varying number of connected peers.

at every iteration. The running time of Average is clearly the fastest, followed closely by Median, Krum, and Bridge. The fast execution of these GARs results from the usage of relatively inexpensive operations, such as calculating the distance between models and sorting the parameters of each dimension. Mozi is relatively slow because it has to evaluate the accuracy of each model. Bristle is initially by far the slowest GAR caused by the performance-based integrator that evaluates the performance not just for each model but for each familiar class of each model. However, when the number of connected peers exceeds β (see Section 3.2), adding more peers has a negligible impact on the speed of Bristle because the distance-based prioritizer is efficient in reducing the number of models to β.

All baselines consume an equal amount of network traffic (roughly 3 MB per iteration per peer for our neural network), but because Bristle transmits only the output layer (roughly 300 KB), the bandwidth requirements of Bristle are reduced by 90%.

6 Related work

In the last five years, a considerable amount of literature has been devoted to creating more effective FL systems. Most work focuses on achieving Byzantine-resilience but also introduces several methods to reduce the bandwidth requirements and to improve learning on non-i.i.d. classes.

Byzantine-resilience. Several well-known Byzantine-resilient GARs are Coordinate-wise Median (CM) [57] which takes the median across all models for each parameter, Krum [6] which selects the model that most closely resembles all other models, and Bulyan [56] which iteratively applies Krum followed by a variant of CM. However, these GARs use distances as a proxy for benignness, which it not reliable as we have seen in Section 3.2. In contrast, performance-based methods reject or accept received models based on their performance on a test dataset. Examples include RONI (Reject On Negative Influence) [5] which simply discards all models with a negative impact on the model, and Zeno [82] / Zeno++ [83] which use a central oracle to estimate the true gradient and only keep the gradients most similar to this estimation. Bristle uses distances only to prioritize the received models and uses per-class performance measurements to integrate models.

Communication efficiency. Because of the bandwidth limitations of cellular networks, numerous methods were proposed to improve the communication efficiency of FL. A popular method is to quantize the gradients to low-precision values [3, 21, 80] or only to transmit the most important parameters (sparse matrix methods) [40, 69]. Bristle only updates the output layer, which works well together with existing techniques to reduce the communication overhead.

Non-i.i.d. learning. Several methods enable learning on non-i.i.d. classes, such as by sharing a small i.i.d. training dataset across all peers [93] or by reusing non-federated continual learning techniques [30, 41, 43, 89]. Bristle also lends several concepts from a non-federated continual learning technique, namely CWR* [53] (see Section 3.3).

FL systems. Several FL systems try to combine Byzantine-resilience with learning on non-i.i.d. data. RSA [47] uses a regularization-based strategy and although it performs relatively well when the data is non-i.i.d., it fares poorly against Byzantine attackers. FoolsGold [29] detects and rejects attacks executed by multiple sybils working together and works well with non-i.i.d. data. However, Zhao et al. [92] show that the Byzantine-resilience of FoolsGold is quite limited. FLeet [20] uses past observed staleness values and similarities with past learning tasks to achieve learning on non-i.i.d. data for a specific type of "soft" Byzantine attacks (namely the presence of stale models) but is unsuitable for other types.

7 Concluding Remarks

We have presented Bristle, middleware for decentralized, federated learning that tolerates Byzantine attacks, even when the classes of the training data are non-i.i.d. By leveraging deep transfer learning, Bristle achieves a high convergence rate despite having a low communication overhead. Through the combination of a fast distance-based prioritizer and a per-class performance integrator, Bristle is able to withstand attacks targeted at subverting the model accuracy. Our experimental evaluation using the popular MNIST dataset has demonstrated these desirable properties and shows that Bristle exhibits superior performance compared to related solutions. In this evaluation, we did not focus on privacy, communication compression, or other aspects that are supplementary to Bristle and addressed in the existing literature. We have implemented and open-sourced Bristle on GitHub.

Although we believe that Bristle is a significant step forward for DFL, we identify two directions for further work. First, communication costs could be further reduced by selectively sending parameters to peers with high class overlap. This can be estimated using Private Set Intersection Cardinality (PSI-CA). One can prevent peers from learning too much about the class distribution of others, e.g., by adding noise to the cardinality and by rate-limiting PSI-CA requests. Second, the accuracy could be increased even more when the non-output layers can also be fine-tuned rather than being fixed. Several popular continual learning algorithms are suitable for this purpose, such as LWF [50], AR1 [53], or EWC [39]. However, tuning also the non-output layers requires the transmission of information about these non-output layers, significantly increasing the communication costs. Additionally, it is non-trivial to maintain the Byzantine-resilience property because the parameters in the hidden layers do not correspond directly with the classes to be predicted.

References

[1] [n.d.]. Google Trends - transfer learning popularity worldwide. ([n. d.]). https://trends.google.com/trends/explore?date=all&q=%2Fm%2F0b6vrv

[2] 2017. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[3] Dan Alistarh, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2016. Qsgd: Randomized quantization for communication-optimal stochastic gradient descent. *arXiv preprint arXiv:1610.02132* 1 (2016).

[4] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. [n.d.]. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2938–2948.

[5] Marco Barreno, Blaine Nelson, Anthony D Joseph, and J Doug Tygar. 2010. The security of machine learning. *Machine Learning* 81, 2 (2010), 121–148.

[6] Peva Blanchard, Rachid Guerraoui, and Julien Stainer. [n.d.]. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*. 119–129.

[7] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, and H Brendan McMahan. 2019. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046* (2019).

[8] Léon Bottou. 2012. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*. Springer, 421–436.

[9] Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. 2018. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics* 112 (2018), 59–67.

[10] Olivier Chapelle and Zaıd Harchaoui. 2005. A machine learning approach to conjoint analysis. *Advances in neural information processing systems* 17 (2005), 257–264.

[11] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. 2019. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635* (2019).

[12] Mingzhe Chen, H Vincent Poor, Walid Saad, and Shuguang Cui. 2020. Wireless Communications for Collaborative Federated Learning in the Internet of Things. *arXiv preprint arXiv:2006.02499* (2020).

[13] Mingqing Chen, Ananda Theertha Suresh, Rajiv Mathews, Adeline Wong, Cyril Allauzen, Françoise Beaufays, and Michael Riley. 2019. Federated learning of N-gram language models. *arXiv preprint arXiv:1910.03432* (2019).

[14] Yudong Chen, Lili Su, and Jiaming Xu. 2017. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1, 2 (2017), 1–25.

[15] Zheyi Chen, Pu Tian, Weixian Liao, and Wei Yu. 2020. Zero Knowledge Clustering Based Adversarial Mitigation in Heterogeneous Federated Learning. *IEEE Transactions on Network Science and Engineering* (2020).

[16] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics* 21, 1 (2020), 1–13.

[17] Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, Pieter Nel, and Sankalp Malhotra. 2018. Notes from the AI frontier: Insights from hundreds of use cases. *McKinsey Global Institute* (2018).

[18] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. 2017. EMNIST: Extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2921–2926.

[19] Georgios Damaskinos, Rachid Guerraoui, Anne-Marie Kermarrec, Vlad Nitu, Rhicheek Patra, and Francois Taiani. 2020. Fleet: Online federated learning via staleness awareness and performance prediction. In *Proceedings of the 21st International Middleware Conference*. 163–177.

[20] Georgios Damaskinos, Rachid Guerraoui, Anne-Marie Kermarrec, Vlad Nitu, Rhicheek Patra, and Francois Taiani. 2020. Fleet: Online federated learning via staleness awareness and performance prediction. In *Proceedings of the 21st International Middleware Conference*. 163–177.

[21] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. [n.d.]. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 561–574.

[22] Roel Dobbe, David Fridovich-Keil, and Claire Tomlin. [n.d.]. Fully decentralized policies for multi-agent systems: An information theoretic approach. In *Advances in Neural Information Processing Systems*. 2941–2950.

[23] John R Douceur. [n.d.]. The sybil attack. In *International workshop on peer-to-peer systems*. Springer, 251–260.

[24] Dave Durkee. 2010. Why cloud computing will never be free. *Commun. ACM* 53, 5 (2010), 62–69.

[25] Markus Endler, Alexandre Skyrme, Daniel Schuster, and Thomas Springer. 2011. Defining situated social context for pervasive social computing. In *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE, 519–524.

[26] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. [n.d.]. Local model poisoning attacks to Byzantine-robust federated learning. In *29th USENIX Security Symposium (USENIX Security 20)*. 1605–1622.

[27] P Felber and Ernst W Biersack. 2004. Self-scaling networks for content distribution. In *Proc. International Workshop on Self-* Properties in Complex Information Systems*. 1–14.

[28] Centers for Disease Control, Prevention, et al. 2003. HIPAA privacy rule and public health. Guidance from CDC and the US Department of Health and Human Services. *MMWR: Morbidity and mortality weekly report* 52, Suppl 1 (2003), 1–17.

[29] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. 2018. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866* (2018).

[30] Camila Gonzalez, Georgios Sakas, and Anirban Mukhopadhyay. 2020. What is Wrong with Continual Learning in Medical Image Segmentation? *arXiv preprint arXiv:2010.11008* (2020).

[31] Shangwei Guo, Tianwei Zhang, Xiaofei Xie, Lei Ma, Tao Xiang, and Yang Liu. 2020. Towards Byzantine-resilient Learning in Decentralized Systems. *arXiv preprint arXiv:2002.08569* (2020).

[32] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[34] István Hegedűs, Gábor Danner, and Márk Jelasity. 2021. Decentralized learning works: An empirical comparison of gossip learning and federated learning. *J. Parallel and Distrib. Comput.* 148 (2021), 109–124.

[35] Hanpeng Hu, Dan Wang, and Chuan Wu. 2020. Distributed machine learning through heterogeneous edge systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 7179–7186.

[36] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[37] Tinu Theckel Joy, Santu Rana, Sunil Gupta, and Svetha Venkatesh. 2019. A flexible transfer learning framework for Bayesian optimization with convergence guarantee. *Expert Systems with Applications* 115 (2019), 656–672.

[38] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, and Rachel Cummings. 2019. Advances and open

problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[39] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017), 3521–3526.

[40] Jakub Konečný and Peter Richtárik. 2018. Randomized distributed mean estimation: Accuracy vs. communication. *Frontiers in Applied Mathematics and Statistics* 4 (2018), 62.

[41] Kavya Kopparapu and Eric Lin. 2020. FedFMC: Sequential Efficient Federated Learning on Non-iid Data. *arXiv preprint arXiv:2006.10937* (2020).

[42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012), 1097–1105.

[43] Swaraj Kumar, Sandipan Dutta, Shaurya Chatturvedi, and MPS Bhatia. [n.d.]. Strategies for Enhancing Training and Privacy in Blockchain Enabled Federated Learning. In *2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM)*. IEEE, 333–340.

[44] Anusha Lalitha, Xinghan Wang, Osman Kilinc, Yongxi Lu, Tara Javidi, and Farinaz Koushanfar. 2019. Decentralized bayesian learning over graphs. *arXiv preprint arXiv:1905.10466* (2019).

[45] Yann Lecun. [n.d.]. http://yann.lecun.com/exdb/mnist

[46] He Li, Kaoru Ota, and Mianxiong Dong. 2018. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE network* 32, 1 (2018), 96–101.

[47] Liping Li, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling. [n.d.]. RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 1544–1551.

[48] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. 2020. Learning to Detect Malicious Clients for Robust Federated Learning. *arXiv preprint arXiv:2002.00211* (2020).

[49] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.

[50] Zhizhong Li and Derek Hoiem. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence* 40, 12 (2017), 2935–2947.

[51] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. [n.d.]. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*. 5330–5340.

[52] Vincenzo Lomonaco. 2019. Continual Learning with Deep Architectures. (2019).

[53] Vincenzo Lomonaco, Davide Maltoni, and Lorenzo Pellegrini. 2019. Rehearsal-Free Continual Learning over Small Non-IID Batches. *arXiv preprint arXiv:1907.03799* (2019).

[54] Songtao Lu, Yawen Zhang, and Yunlong Wang. [n.d.]. Decentralized federated learning for electronic health records. In *2020 54th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 1–5.

[55] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. [n.d.]. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*. PMLR, 1273–1282.

[56] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. 2018. The hidden vulnerability of distributed learning in byzantium. *arXiv preprint arXiv:1802.07927* (2018).

[57] Luis Muñoz-González, Kenneth T Co, and Emil C Lupu. 2019. Byzantine-robust federated machine learning through adaptive model averaging. *arXiv preprint arXiv:1909.05125* (2019).

[58] Angelia Nedic and Asuman Ozdaglar. 2009. Distributed subgradient methods for multi-agent optimization. *IEEE Trans. Automat. Control* 54, 1 (2009), 48–61.

[59] Solmaz Niknam, Harpreet S Dhillon, and Jeffrey H Reed. 2020. Federated learning for wireless communications: Motivation, opportunities, and challenges. *IEEE Communications Magazine* 58, 6 (2020), 46–51.

[60] Jialin Pan. 2010. *Feature-based transfer learning with real-world applications*. Ph.D. Dissertation. Citeseer.

[61] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett Landman, and Klaus Maier-Hein. 2020. The future of digital health with federated learning. *arXiv preprint arXiv:2003.08119* (2020).

[62] Felix Sattler, Thomas Wiegand, and Wojciech Samek. 2020. Trends and advancements in deep neural network communication. *arXiv preprint arXiv:2003.03320* (2020).

[63] William Schneble and Geethapriya Thamilarasu. [n.d.]. Attack Detection Using Federated Learning in Medical Cyber-Physical Systems. ([n. d.]).

[64] Khe Chai Sim, Françoise Beaufays, Arnaud Benard, Dhruv Guliani, Andreas Kabel, Nikhil Khare, Tamar Lucassen, Petr Zadrazil, Harry Zhang, and Leif Johnson. [n.d.]. Personalization of end-to-end speech recognition on mobile devices for named entities. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 23–30.

[65] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. [n.d.]. Federated multi-task learning. In *Advances in Neural Information Processing Systems*. 4424–4434.

[66] Konstantin Sozinov, Vladimir Vlassov, and Sarunas Girdzijauskas. [n.d.]. Human Activity Recognition Using Federated Learning. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*. IEEE, 1103–1111.

[67] Quinten Stokkink, Dick Epema, and Johan Pouwelse. 2020. A Truly Self-Sovereign Identity System. *arXiv preprint arXiv:2007.00415* (2020).

[68] Quinten Stokkink and Johan Pouwelse. [n.d.]. Deployment of a blockchain-based self-sovereign identity. In *2018 IEEE international conference on Internet of Things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)*. IEEE, 1336–1342.

[69] Nikko Strom. [n.d.]. Scalable distributed DNN training using commodity GPU cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*.

[70] Lili Su and Nitin H Vaidya. 2015. Fault-tolerant distributed optimization (Part IV): Constrained optimization with arbitrary directed networks. *arXiv preprint arXiv:1511.01821* (2015).

[71] Shreyas Sundaram and Bahman Gharesifard. 2018. Distributed optimization under adversarial nodes. *IEEE Trans. Automat. Control* 64, 3 (2018), 1063–1076.

[72] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. 2018. A survey on deep transfer learning. In *International conference on artificial neural networks*. Springer, 270–279.

[73] Hanlin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu. [n.d.]. Communication compression for decentralized training. In *Advances in Neural Information Processing Systems*. 7652–7662.

[74] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. [n.d.]. Data poisoning attacks against federated learning systems. In *European Symposium on Research in Computer Security*. Springer, 480–501.

[75] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* (2017).

[76] Dan S Wallach. 2002. A survey of peer-to-peer security issues. In *International symposium on software security*. Springer, 42–57.

[77] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. 2019. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications* 37, 6 (2019), 1205–1221.

[78] Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. 2020. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 22, 2 (2020), 869–904.

[79] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. [n.d.]. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2512–2520.

[80] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *arXiv preprint arXiv:1705.07878* (2017).

[81] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. 2018. Generalized byzantine-tolerant sgd. *arXiv preprint arXiv:1802.10116* (2018).

[82] Cong Xie, Sanmi Koyejo, and Indranil Gupta. [n.d.]. Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In *International Conference on Machine Learning*. PMLR, 6893–6901.

[83] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Zeno++: Robust Fully Asynchronous SGD. *arXiv preprint arXiv:1903.07020* (2019).

[84] Xiaofei Xie, Lei Ma, Haijun Wang, Yuekang Li, Yang Liu, and Xiaohong Li. [n.d.]. DiffChaser: Detecting Disagreements for Deep Neural Networks. In *IJCAI*. 5772–5778.

[85] Zhixiong Yang and Waheed U Bajwa. 2019. BRIDGE: Byzantine-resilient decentralized gradient descent. *arXiv preprint arXiv:1908.08098* (2019).

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

[87] Zhixiong Yang, Arpita Gang, and Waheed U Bajwa. 2019. Adversary-resilient inference and machine learning: From distributed to decentralized. *stat* 1050 (2019), 23.

[88] Zhixiong Yang, Arpita Gang, and Waheed U Bajwa. 2020. Adversary-resilient distributed and decentralized statistical inference and machine learning: An overview of recent advances under the Byzantine threat model. *IEEE Signal Processing Magazine* 37, 3 (2020), 146–159.

[89] Xin Yao and Lifeng Sun. [n.d.]. Continual Local Training For Better Initialization Of Federated Models. In *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 1736–1740.

[90] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. 2018. Byzantine-robust distributed learning: Towards optimal statistical rates. *arXiv preprint arXiv:1803.01498* (2018).

[91] Yu Zhang and Qiang Yang. 2017. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114* (2017).

[92] Lingchen Zhao, Shengshan Hu, Qian Wang, Jianlin Jiang, Shen Chao, Xiangyang Luo, and Pengfei Hu. 2020. Shielding Collaborative Learning: Mitigating Poisoning Attacks through Client-Side Detection. *IEEE Transactions on Dependable and Secure Computing* (2020).

[93] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582* (2018).

[94] Sicong Zhou, Huawei Huang, Wuhui Chen, Pan Zhou, Zibin Zheng, and Song Guo. 2020. Pirate: A blockchain-based secure framework of distributed machine learning in 5g networks. *IEEE Network* (2020).

Supplementary Material

This supplementary material complements the thesis article with additional information relevant to the thesis committee, similar to several other thesises published in the EEMCS faculty such as [1] and [2]. The supplementary material is composed of four main sections: extended preliminaries, related work, implementation, and discussion. In the extended preliminaries, we aim to give the thesis committee members a better understanding of the basic mechanisms behind machine learning and federated learning. In the extended related work, we devote significant attention to explaining how current federated learning technologies aim to solve the challenges related to Byzantine-resilience, non-i.i.d. data, and communication efficiency. In the extended implementation section, we give more details about the implementation of the main contribution and about the setup of the test environment. Finally, in the extended discussion, we discuss in detail the weaker points of the algorithm and give a large number of directions for further research. Additionally, we enumerate the biggest issues that we encountered during the research.

[1] https://repository.tudelft.nl/islandora/object/uuid%3A2e793ece-4572-4bb6-83e3-541be467cb4f
[2] https://repository.tudelft.nl/islandora/object/uuid%3A6c104c80-aef6-4657-b9f5-6683a0e45b14

A

Preliminaries to Federated Learning

Because Federated Learning (FL) is a relatively new scientific field, we aim to give the reader in this section a better understanding of the basic idea behind FL. In the next section, we will zoom in on more advanced FL systems.

A.1. Basic idea behind federated learning

In traditional machine learning, we aim to minimize the global cost function, risk function, loss function, or score $\ell(\theta)$ by finding the optimal model θ^*:

$$\theta^* = \underset{\theta}{argmin} \underset{(x,\, y) \in D}{\mathbb{E}} \ell\left(f_\theta(x),\, y\right) \tag{A.1}$$

In this equation, θ is the model, D is a distribution over X x Υ (with X denoting the data and Υ denoting the corresponding labels), and $\ell(\theta, i)$ is the loss of model θ on dataset instance i. This loss function is a proxy for the actual error to be minimized and is usually the negative log-likelihood of the ground truth class in a classification problem.

This optimization problem is known as risk minimization, but unfortunately solving this problem is intractable for more complex models. Therefore, a technique called Empirical Risk Minimization (ERM) is commonly used where a dataset M is sampled i.i.d. from D. The optimal model can then be estimated by calculating:

$$\theta_n = \underset{\theta}{argmin} \frac{1}{|M|} \sum_{(x,\, y) \in M} \ell\left(f_\theta(x),\, y\right) \tag{A.2}$$

A popular technique to optimize this function is called Gradient Descent (GD) which iteratively takes the derivative of the loss function with respect to the training samples and then moves the hyperparameters in the direction of the gradient:

$$\theta^{t+1} = \theta^t - \lambda \nabla_\theta \ell(\theta, i) \tag{A.3}$$

However, because the dataset can be large, it can take a long time for gradient descent to converge. A faster approach, used by almost all ML algorithms today, is to use Stochastic Gradient Descent (SGD), [1] where a subset (a *minibatch*) of the dataset is selected to update the parameters [2, 3]. As a result, SGD produces faster but noisier updates than GD, but this noise is not necessarily a drawback as it also helps the algorithm escape local minima. An important requirement for SGD to converge is that each minibatch is an unbiased sample of the true distribution, usually achieved through uniform random sampling [4].

The most straightforward way to apply stochastic gradient descent in a distributed or federated setting is to use a single server (called the *parameter server*) and several clients. The parameter server distributes and aggregates the global model $w = \bigcup_{i \in \mathcal{N}} w_i$, and each client i trains the model that it obtained from the server before sending back the result w_i [5, 6].

The most trivial implementation of stochastic gradient descent in federated learning systems is called FedAvg [7]. FedAvg aggregates the models owned by the peers by using coordinate-wise weighted averaging. It was introduced by Google and is still extensively researched from both an applied and theoretical perspective [8]. For non-convex parameter spaces, averaging models can yield poor results since good models can converge in different directions, but neural networks are mostly convex when they are sufficiently over-parameterized and therefore not very prone to bad local minima [9–11]. The pseudocode for FedAvg is given in Algorithm 1.

Algorithm 1 FedAvg: basic centralized federated learning executed on a single server and a set of clients

```
1: Input
2:     B    local minibatch size
3:     K    set of nodes
4:     m    number of nodes per iteration
5:     E    #local epochs
6:     $\eta$    learning rate
7:     D    set of local datasets
8: procedure SERVER
9:     Initialize $w$
10:    for each round $t = 0, 1, 2, \dots$ do
11:        $S[t]$ ← (random set of $m$ clients from $K$)
12:        for each client $k$ in $S[t]$ in parallel do
13:            $w'[k][t+1] \leftarrow \text{ClientUpdate}[k][w[t]]$
14:        end for
15:        $w[t+1] \leftarrow \frac{1}{\sum_{k \in S[t]} |D[k]|} \sum_{k \in S[t]} |D[k]| * w'[k][t+1]$
16:    end for
17: end procedure
18: procedure CLIENT(k)
19:    $\mathcal{B}[k]$ ← (split $D[k]$ into batches of size $B$)
20:    for each local epoch e=0, 1, ..., E do
21:        for batch $b$ in $\mathcal{B}[k]$ do
22:            $w \leftarrow w - \eta \nabla l(w; b)$
23:        end for
24:    end for
25:    return $w$ to server
26: end procedure
```

B RELATED WORK

Over the past five years, a significant amount of articles have been written about tackling the major obstacles to wide-scale adoption of FL. Three important challenges are - as explained in the Introduction section of the main article - the mitigation of Byzantine attacks, the ability to handle non-i.i.d. data, and the reduction of the communication costs. In this section we conduct a thorough literature review and investigate existing methods proposed to tackle these challenges.

B.1. BYZANTINE-ATTACKS

A very popular (and simple) federated learning technique to combine the model vectors is to simply take their average. Obviously, when a single Byzantine node transmits a model with extremely low or high values, the average significantly changes, and the model become worthless. Such an attack is easy to detect, but there are many more sophisticated attacks that, even with a single Byzantine attacker, can considerably reduce the model's performance and are much more difficult to detect[12].

Byzantine attacks can be classified into data poisoning or model poisoning attacks, and untargeted or targeted attacks as shown in Table B.1.

Untargeted	Data poisoning	[13–18]
	Model poisoning	[19, 20]
Targeted	Data poisoning	[14, 17, 21–29]
	Model poisoning	[28, 30–35]

Table B.1: Classification of attacks

B.1.1. DATA POISONING VS MODEL POISONING ATTACKS

Data poisoning attacks such as [13–18, 21–29] train the model with dirty samples to subvert the performance of the learned model to such an extent that the model becomes worthless. They were introduced by Gu et al. [22] to attack support vector machines and later extended to many other ML algorithms including neural networks. The most popular data poisoning attack is a label-flip attack, where the labels of two or more classes are changed [14, 17, 30]. One might think that sending completely random numbers would also be an effective attack. However, because the mean of completely random numbers is 0, the network will still converge when the standard deviation is not too extreme [36] (in fact, adding noise to the parameters is a popular method called differential privacy that is used to improve the user's privacy[37–39]).

While data poisoning attacks are based on the manipulation of training data, model poisoning attacks (introduced by [30]) such as [19, 20, 28, 30–35] manipulate the model's parameters before sharing the updated model with other nodes. Consequently, every data poisoning attack can be imitated with a model poisoning attack [40] because manipulating the training data is only a means to manipulate the model's parameters. A simple model poisoning attack is the Gaussian attack, where some of the gradient vectors are replaced by random vectors sampled from a Gaussian distribution with large variances. Because model poisoning attacks give the attacker full control over every single parameter, they can be much more effective as recent research has shown [12, 33, 41]. More sophisticated attacks can even be used to replace the entire global model with a model of the attacker's choice (model replacement attack), given a carefully chosen scaling factor [30].

B.1.2. UNTARGETED VS TARGETED ATTACKS

Another way to classify Byzantine attacks is to group them into untargeted and targeted attacks[40]. Whereas untargeted attacks (also known as convergence-prevention attacks [31] or poisoning availability attacks [42]) such as [13–20, 41] aim to prevent convergence and reduce the global model's accuracy [20, 41], targeted attacks (also known as (semantic) backdoors, Trojan threats, stealth attacks, or poisoning integrity attacks [42]) such as [14, 17, 21–35] aim to alter the model's behavior in specific situations while keeping the total accuracy as high as possible to mislead Byzantine-defense mechanisms [30, 33]. Without proper defense mechanisms, federated learning is susceptible to both untargeted and targeted attacks [14].

In targeted attacks the attacker aims to manipulate the model so that it misclassifies only certain classes. Popular examples are a label-flip attack or a trigger attack (which is based on an almost invisible attacker-chosen pattern of pixels). A particularly effective attack is described by Bhagoji et al. [32] who use an alternating minimization strategy (alternately minimizing training loss and boosting specific parameters for the malicious objective). A more sophisticated attack is proposed by Xie et al. in [28] who note that all backdoor attacks until then used embeddings of the same global trigger pattern for all Byzantine parties. They then propose distributed backdoor attacks (DBA) where the global trigger pattern is decomposed into local patterns which is then embedded in different Byzantine parties, thus making the attack harder to detect, easier to bypass robust aggregation rules, and being more effective. In line with this contribution, Baruch et al. [31] show that targeted model poisoning attacks can become both significantly more effective and harder to detect when adversaries are able to collude.

Backdoors are a specific type of targeted attacks where the accuracy of the model is not impacted for any benign sample, but only for samples manipulated with a specific pattern that only the attacker knows. Detecting these backdoors is an NP-hard problem by a reduction from 3-SAT [43] and unlikely to be detected using gradient based techniques. To illustrate this, Wang et al. [43] explain how it is relatively easy to develop a so-called edge-case backdoor which forces a model to consistently misclassify seemingly easy inputs that are unlikely to be part of the regular training data. Because these backdoors only need to modify a small part of the model [40], they look quite similar to benign updates and require fewer adversaries than untargeted attacks.

B.2. BYZANTINE-RESILIENCE

In this section, we provide an extensive overview of Byzantine-resilient defense methods (since the main contribution of this thesis is also a Byzantine-resilient defense method) and categorize these methods into five distinct categories.

Byzantine resilience can be divided into weak and strong Byzantine resilience [44]. Weak f-Byzantine resilience implies that despite the presence of f Byzantine nodes, the network will almost certainly converge to some value. Strong f-Byzantine resilience implies that the network not only converges in the presence of f Byzantine nodes, but also converges to approximately the right value. In this thesis we will focus on strong Byzantine resilience.

An interesting observation made by Haykin [45] is that a “mild” Byzantine worker can actually improve the performance of the system. This has to do with the fact that the optimization function of a neural network is often not entirely convex but has many local optima. By providing the “wrong” direction, a little bit of noise (or a “mild” Byzantine attack in that regard) can pull the optimization function out of a local minimum so that the network can converge to a better global minimum [46–48]. This is also the reason why SGD works so well: a randomly drawn sample is inherently more noisy (higher variance) than the average of all samples [49] and may pull the network out of a local minimum. However, stronger Byzantine attacks can pull the network away from the global minimum in which case they subvert the network's performance.

There are several types of Byzantine-resilient defense mechanisms (usually referred to as Gradient Aggregation Rules (GARs) in the literature) that are often segmented into distance-based GARs and performance-based GARs. Distance-based GARs are based on the calculation of some kind of distance between potential malicious attack vectors and some other vector(s). They are usually efficient but also vulnerable to elaborately designed Byzantine attacks [31, 41]). Performance-based GARs are based on testing the accuracy of a potentially malicious model on a small representative dataset. They are usually computationally quite expensive, depend on the availability of a test dataset, and are usually quite effective [50]. Other ways of segmenting these algorithms is whether they are centralized (dependent on the availability of a single server), by their degree of dimensional Byzantine resilience [50] (namely, the maximum number of tolerated Byzantine workers), their ability to handle non-i.i.d. data, and their ability to perform well in asynchronous settings. The most notable GARs that we will discuss are compared based on these aspects in Table B.2.

B.2.1. DISTANCE-BASED SCREENING

Screening the distance of potentially malicious incoming model updates to the peer's own trusted model is by far the most popular method to evade Byzantine attacks, which should come as no surprise. Measuring the distance is relatively efficient, does not depend on an additional dataset, special hardware features, or an additional server, and it provides excellent protection against relatively simple attacks. However, although this class of algorithms is effective against simple attacks such as Gaussian noise and label-flip attacks, the performance is bad when more advanced attacks are used[51]. This is due to an implicit and somewhat erroneous assumption of distanced-based GARs, namely that short distances between model parameters imply comparable performance. Additionally, when the data is non-i.i.d. between peers, the distance between benign gradient updates can be large, resulting in the rejection of these updates by distance-based GARs. Moreover, when the peer's own model is extremely stale, all incoming models that are up-to-date have a high distance and are thus considered outliers, making it hard for the peer to catch up.

Krum [52] is a particularly influential algorithm which selects the model that most closely resembles (in terms of Euclidean distance) all other models as the new global model. Even if the selected model is malicious, the performance should not degrade too much in theory since this model is relatively close to all other models. Despite theoretical guarantees for the convergence for certain objective functions, Krum appears to perform poorly compared to other algorithms [53] and often converges to an ineffective model [20]. The deficient performance stems from the ability of Byzantine workers to make a substantial change in a single parameter without significantly influencing the total distance due to the typically high dimensionality of the parameters [20]. Baruch et al. [31] elaborate upon this insight and argue that since only a single model is selected and even the best benign worker will have a few parameters deviated far from the mean, Krum performs worse than other GARs that integrate data from multiple models into the final model. The authors also briefly discuss Multi-Krum, which achieves comparable accuracy at a faster rate by using the average of a number of local gradients obtained by Krum.

CTM / CM [12, 54, 55] are two simple distance-based GARs. Coordinate-wise Trimmed Mean (CTM) simply cuts off the smallest and largest b values in each dimension of the incoming vectors and Coordinate-wise Median (CM) takes the median in each dimension. CTM needs at least $2b+1$ models where b is the maximum number of attackers. CM does not depend on an a-priori specified number of attackers, but does incur a performance hit because each dimension must be sorted to get the median.

GeoMed / MeaMed [50] were compared by Xie et al. [50] under non-convex settings with Krum. The Geometric Median(GeoMed) is defined as [50, 52, 56]:

$$\underset{v \in \mathbb{R}^d}{argmin} \sum_{i=1}^{n} \| v - \tilde{v}_i \| \tag{B.1}$$

This formula can be interpreted as the point for which the square distance to all other points in an n-dimensional space is minimized. The Mean around the Median (MeaMed) is defined as the mean value of the $n-q$ indices closest to the median, where q is an arbitrary value. The authors find that Krum, Multi-Krum, and the Geometric Median perform worst, the Marginal Median has considerable variance, and the Mean around the Median performs best. The Geometric median not only performs poorly, but also dominates the training time in large-scale settings [57].

(Geometric) Median of Means [56, 58–62] is a variant on the Geometric Median that first partitions all received vectors into k batches, then computes the mean for each batch, and finally takes the geometric median of the k batch means. Chen et al. [56] extend the techniques described in [62] with arbitrary/adversarial outliers. However, their algorithm fails even when there is only a single Byzantine node in each mini-batch and is thus not reliable.

Bulyan [20] combines the strengths of Krum and CM by iteratively applying Krum to select a number of models followed by a variant of CM. More specifically, Bulyan finds for every dimension the n parameters closest to the median and then takes their mean value. A notable disadvantage of Bulyan is its speed and the stringent condition that it imposes on the number of Byzantine nodes, namely $\#nodes \geq 4\ x\ \#nodes_{byzantine} + 3$. A year later, the authors extend Bulyan to Multi-Bulyan, in the same way as the extension of Krum to Multi-Krum, but unfortunately they did not report the results [44].

SignSGD [63] was developed to reduce the network traffic resulting from the aforementioned GARs by transmitting only the sign of each dimension of the gradient at each iteration. Since the global model is updated with an element-wise majority vote on the signs of the received gradients, the algorithm is in fact a median-based algorithm that also makes it robust against certain Byzantine attacks and guarantees convergence when the noise behaves along certain conditions [64]. However, one of these conditions is that the data is i.i.d., which is typically not the case in federated learning environments[65]. Sohn et al. make SignSGD more robust against MITM-attacks, but do not address the case where nodes themselves are malicious[66].

RSA [67], confusingly having the same name as the well-known cryptosystem, is in contrast to the aforementioned methods able to handle non-i.i.d. data in a setting with multiple adversaries. It aims to prevent incorrect gradient aggregation by letting every node store and update a local version of the global model. These local version are then aggregated by the server by means of an ℓ_p-norm regularization term regularizing the magnitudes of malicious messages. RSA is somewhat non-i.i.d.-resilient: it performs significantly better than Krum and median-based methods but achieves even in a mildly Byzantine non-i.i.d. environment an accuracy of just 56% on MNIST. All GARs described until now assume a federated setting where a single parameter server iteratively updates the global model. However, these algorithms do not translate well into a decentralized setting (which is the focus of this thesis) because decentralized GARs require consensus between all peers which is usually not required for distributed learning. To the best of our knowledge, there are only three papers that attempt to achieve Byzantine-resilient decentralized learning by adopting a truly distance-based strategy, namely ByRDiE [68], BRIDGE [69], and some extension of BRIDGE [70].

ByRDiE [68] recognizes that regular CM algorithms are suboptimal in vector-valued problems, because simply minimizing the objective function along each coordinate independent of all other coordinates yields the wrong solution (unless all dimensions are truly independent, which is generally not the case). The authors overcome this limitation by cyclically updating every coordinate one by one in a decentralized manner and subsequently applying trimmed-mean screening to obtain the final coordinate for each dimension. Given a strictly convex loss function, ByRDiE is proven to always converge (although not necessarily towards an optimal solution). However, although ByRDiE might be efficient in terms of required training samples, it is inefficient in terms of network communication because it only updates one coordinate at a time [53]

BRIDGE [69] aims to significantly reduce the network communication for high dimensional problems and still achieve decentralized Byzantine-resilience by combining CTM with SGD. The same authors later showed better performance for BRIDGE than for CTM [53], which is surprising because BRIDGE boils down to CTM in a distributed environment. Upon closer examination, this happens because the authors use a 0.7 connection ratio between the nodes to evaluate BRIDGE and only (the authors conveniently omitted this number so that the reader has to calculate it himself) a $4 \times \#max\ byzantine\ nodes + 1 = 4 \times 2 + 1 = 9; \quad 9 / 20\ nodes = 0.45$ connection ratio between the nodes to evaluate CTM. Peng et al. [70] show that BRIDGE's ability to handle non-i.i.d. data can be improved by adding a total variation (TV) norm penalty to allow some outliers. This likely reduces the ability of the algorithm to defend against noise attacks, but unfortunately the authors omitted these results from the paper. He et al. [71] also extend BRIDGE to non-i.i.d. settings, but do this by re-introducing the central server that we wanted to omit in a decentralized setting in the first place.

One of the most recent articles about this topic is [72] which select the models with the smallest Euclidean norm to be averaged for the updated model, but for some reason the authors decided to evaluate its performance by measuring the loss function. The loss function of neural networks is extremely noisy and also relatively unreliable compared to an evaluation using a validation dataset, thus making it hard to properly estimate its performance.

An interesting distance-based method is described in [73] where the authors construct a graph where the nodes (representing models) are connected by a vertex only when their Euclidean distance is small enough, and subsequently solve the maximum clique model to find the set of models that are similar to each other and therefore probably benign. Unfortunately, the authors only evaluate trivial label-flip attacks, so it is unclear how effective the algorithm is in a more challenging environment.

B.2.2. PERFORMANCE SCREENING

Although distance-based screening methods can be quick and effective to filter out "unusual" models, they will not include benign models when these models are quite different from the other models (e.g., when the data is non-i.i.d. or the peer's own model is very stale) and also allow an attacker to let the model drift towards a bad solution. Performance-based solution such as [27, 74–78] detect malicious models based on their impact on the model's accuracy given a test dataset. A major advantage of performance-based solutions is that, whereas many other GARS assume that the number of adversarial workers is always less than half of the total number of workers, performance-based solutions typically ensure convergence even when the number of adversaries exceeds the number of benign nodes [76, 78–82]. From all these papers, we want to seriously criticize the paper written by Zhao et al. [78] because, aside from the fact that it contains serious grammar errors and completely incorrect references, it also includes a major error about when label-flipping attacks are preferred above backdoor attacks. The authors say that label-flipping attacks are more effective in a scenario where data samples with the same label are quite similar while the latter is more suitable for scenarios where samples with the same label are quite diverse. This is incorrect: you want to use label-flipping attacks as an effective way to fool or prevent convergence of a model without any serious byzantine-resilient GAR while you want to use backdoor attacks to trick the model to misclassify certain input data without letting anyone notice that you are malicious (see Section B.1.2). The authors also assume that agents share which labels they own, which is absurd: the whole purpose of a federated learning environment is to keep the user's data (including the labels) private.

RONI [74] / **TRIM** [42] are the most basic types of performance-based GARs. RONI (Reject On Negative Influence) removes training examples with a negative impact on the accuracy of the model. TRIM finds a subset of the training dataset given a pre-specified size and set of hyperparameters that maximize the accuracy. TRIM is, at least according to the authors, more effective than RONI. Both methods were originally intended to filter out bad training data on a single node, but Fang et al. [41] converted and applied RONI and TRIM to a federated setting and found that in a federated setting RONI gives slightly better performance.

Zeno [83] / **Zeno++** [82] were introduced by Xie et al. for synchronous environments and asynchronous environments respectively. Both use a centralized oracle that estimates based on a validation dataset the true gradient and only keeps the gradients most similar to this estimation. The performance of both GARs is quite good according to Yang et al. [53], but they depend on a centralized parameter server and need access to a sufficiently large unbiased validation dataset.

PDGAN [84] works quite differently compared to the other approaches and uses a Generate Adversarial Network (GAN) to reconstruct the training data used by the peers to train the network. Based on this data, the accuracy of the received models can be estimated reliably after a large number of iterations (needed to train the GAN). However, since the training data used by the peers is supposed to stay private, it is actually quite disturbing that GANs are able to reconstruct this data [32, 85], and are, in that regard, also a "highly impactful and prioritized" [86] attack in their own right.

Mozi [51] was an important inspiration for this thesis and first applies a distance-based strategy to quickly select a candidate pool of probably benign nodes, and then screens the resulting nodes based on their performance on a test dataset (performance screening).

B.2.3. PRUNING

Since backdoor attacks (see Section B.1.2) are extremely challenging to detect, an entirely different class of GARs called "pruning" defenses has been proposed, specifically aimed at preventing these backdoor attacks [87–89]. Pruning defenses use a representative subset of the global dataset (partially violating the FL assumption [40]) to evaluate which neurons in the neural network are inactive. These neurons are important to find and subsequently remove because they enable attackers to create a backdoor in the model [22].

Unfortunately, even when these inactive neurons are removed from the model, more sophisticated backdoor attacks are still possible [90]. After all, the boundary between a neuron being unused or being actively used is vague. There are several other methods aimed at detecting backdoors [77, 88, 89, 91–96], but these methods either assume that there is a central server that can access the whole training dataset and scan the samples for malicious samples (which is clearly impossible in a federated learning system) or that there is a holdout set of similarly distributed data available (which cannot help defend against more sophisticated model poisoning attacks as discussed in Section B.1.2).

B.2.4. BEHAVIORAL-BASED

FoolsGold [14] is an algorithm that detects and rejects attacks executed by multiple sybils working together. The authors observed that when sybils collude to poison a model, their "behavior is more similar to each other than the similarity observed amongst the honest clients". However, Zhao et al. [78] showed that FoolsGold is unable to defend against a powerful attack performed by a single node instead of multiple colluding sybils, and can also be evaded by decomposing a distributed attack into several orthogonal vectors.

Whereas all GARs discussed so far attempt to make it as difficult as possible for an attacker to manipulate the system, there is also a wide variety of GARs that take a different approach and aim to eliminate any incentive for a node to attack the system. A trivial approach where a parameter server simply assigns a reputation based on a performance-based screening procedure (such as [81]) does not work well, because a Byzantine attacker can first build up an excellent reputation, and then suddenly significantly subvert the model's performance, empowered to do so thanks to its good reputation. A better approach appears to be to reward and punish participants based on their contributions, something that can be facilitated in decentralized environments through a distributed ledger [97–112], usually a blockchain. This ledger can also be used to save global model parameters to enhance the system security [100, 104].

Kang et al. [113] introduced reputation as a means to determine the reliability of each node and subsequently proposed a GAR based on these reliability scores[114], using RONI to calculate reputations when the data is i.i.d., and FoolsGold to calculate reputations when the data is non-i.i.d. . For this to work, the authors (implicitly) assume an environment where nodes have a strong identity so that attackers cannot create a large number of sybils.

Zhao et al. [115] also assign a reputation to nodes that contribute well, but their algorithm is seriously flawed: the authors use KRUM to determine if an update is benign (which is highly unreliable [53]) and then increase / decrease a node's reputation when the update is accepted / rejected respectively, implying that you can make for every good contribution also a bad contribution. However, in practice a single bad contribution can significantly damage the model while the impact of a single good contribution on the model is generally quite limited.

Whereas all former approaches assume that the individual workers might be Byzantine, [116] assumes a centralized setting where the parameter server might be Byzantine. They use a blockchain to audit all model updates from all peers so everyone can verify that the parameter server aggregates the model updates correctly. The authors also train an autoencoder to spot outliers (i.e., Byzantine attack). This seems to work fairly well, but the autoencoder is only effective after it has been trained properly which may take many iterations.

Another blockchain-based approach called HoldOut SGD [97] first segments the nodes into a set of workers and a voting committee. The workers use their data to train the model for a single iteration and the voting committee votes for the best proposals and stores this information on a blockchain (similar to [107, 117]; The voting committee is selected on the basis of Proof of Stake (Pos) and Verifiable Random Functions (VRFs)). The method is fully decentralized but requires that the maximum number of Byzantine attackers is limited to 1/3rd of all nodes. The technique is hardly scalable to a large number of nodes, because each node in the voting committee has to evaluate every single update. Additionally, a significant amount of time is spent idling for each node because either all voting committee nodes are waiting for the workers to be finished or vice versa.

Although the blockchain papers mentioned above are of reasonable quality, one has to be very careful when searching for literature about this subject. There are many papers where a blockchain is used for federated learning without understanding its (dis)advantages. For example, Lu et al. [103] say that they want to address privacy issues by using a blockchain, but simply using a blockchain does not magically improve the user's privacy. The authors also state that Directed-Acyclic-Graphs (DAGs) are a certain kind of blockchain (which is incorrect, they are different technologies. Stating that DAGs and blockchains are both examples of Distributed Ledger Technology (DLT) would have been correct) and that DAGs use cumulative Proof of Work (PoW), which is also incorrect: DAGs usually just reference and validate previous transactions without any PoW involved.

A particularly good paper where the authors really take advantage of DLT's strengths is written by Schmid et al. [106]. The authors use a Tangle to represent the approved transactions as nodes in a DAG. For each new transaction, the system first verifies two previous transactions by using a distance-based or performance-based screening procedure. When the transactions are approved, the previous transactions are merged with the current model to represent the updated model parameters.

There is also a considerable body of literature that uses behavioral techniques to incentive nodes with high quality training data to participate in the training process such as [98, 108–111, 113, 114, 118–130] and Stackelberg game methods [121, 130–132], but since these methods are not intended to defend against Byzantine attacks, we leave them out of the scope of this thesis. In addition, the underlying assumption that agents should be given some kind of incentive to participate in a federated training process does not seem to apply in many popular FL applications, such as Gboard, Captcha, or Google Fit.

B.2.5. Other

There are several articles that discuss innovative GARs that are not easy to classify into a particular category. There are a few papers that use Trusted Execution Environments (TEEs) to achieve some form of security, such as [110, 133–136]. Bonawitz et al. use secure aggregation based on the Secure Multi-party Computation (SMC) algorithm to aggregate the values of untrusted nodes without revealing these values, enabling a parameter server that each party can fully trust [133]. Sabt et al. discuss how TEEs can also be used as a defense technique [136], whose insights are later used to create a generic

	GAR	**Condition on [M, b]**	**No prior information about #attackers**	**Ability to learn non-i.i.d. classes**	**Decentralized**
Distance-based	FedAvg[7]	N/A	N/A	✓	✗
	CM[12]	$M \geq 2b+1$	✓	✗	✗
	CTM[12]	$M \geq 2b+1$	✓	✗	✗
	GeoMed[50]	$M \geq 2b+1$	✓	✗	✗
	Krum[52]	$M \geq 2b+3$	✗	✗	✗
	Multi-Krum[52]	$M \geq 2b+m+2$	✗	✗	✗
	Bulyan[20]	$M \geq 4b+3$	✗	✗	✗
	RSA[67]	$M \geq 2b+1$	✓	✓	✗
	SignSGD[63]	$M \geq 2b+1$	✓	✗	✗
	ByRDiE[68]	$M \geq 2b+1$	✗	✗	✓
	BRIDGE[69]	$M \geq 2b+1$	✗	✗	✓
Performance-based	RONI[74]	$M \geq b+1$	✓	✗	✗
	TRIM[42]	$M \geq b+1$	✓	✗	✗
	Zeno[83]	$M \geq b+1$	✗	✗	✗
	Zeno++[82]	$M \geq b+1$	✗	✗	✗
	PDGAN[84]	$M \geq b+1$	✓	✗	✗
	MOZI[51]	$M \geq b+1$	✓	✗	✓
Other	FoolsGold[14]	$M \geq b+1$	✓	✓	✗
	DRACO[57]	$M \geq b+1$	✓	✗	✗
	Bristle (this work)	$M \geq b+1$	✓	✓	✓

Table B.2: Overview of the most notable GARs

framework that can be used to integrate TEEs in a federated learning environment [134, 135]. Weng et al. [110] developed DeepChain that, on the one hand, uses a blockchain to incentivize parties to participate in the training process, and, on the other hand, uses a combination of Intel Software Guard Extensions (SGX) enclaves and homomorphic cryptographic functions to provide a safe and privacy-preserving system. Their solution works well, but is also computationally very expensive, limiting its use cases.

There are also a few solutions that model defending against Byzantine attacks as a learning problem. Ji et al. use a Recurrent Neural Network (RNN) and an auxiliary dataset to aggregate gradients in a Byzantine-resilient manner [80]. The idea is that a machine learning approach can detect attacks that are difficult to detect for other more straightforward algorithms. Unfortunately, since their RNN is a "black box", the authors are unable to give any theoretical guarantees. A year later, the same author uses variational autoencoders with spectral anomaly detection to detect malicious updates based on their low-dimensional embeddings [19]. By removing the noisy and irrelevant features, the anomalous (malicious) model updates can be distinguished from the benign updates in a low-dimensional latent feature space.

DRACO [57] is a well-cited example of a final type of GARs we would like to highlight:

GARs based on replicating the training process over multiple nodes [57, 66, 137, 138]. When nodes are benign, they will report the same results when give the same training data (under several assumptions). While the accuracy of these GARs is often illustrated by rigorous theoretical guarantees, they typically assume a centralized server with either a copy of the data or the ability to globally shuffle the data, making the algorithm inappropriate for a decentralized federated learning environment. DRACO lets the parameter server send the same chunk of data to multiple workers and uses majority voting to find the correct evaluation. When the number of benign nodes is larger than the number of Byzantine nodes, DRACO is very robust, but the algorithm scales poorly to a greater number of attackers. For example, when there are just 5 attackers, each chunk already needs to be calculated $5 \times 2 + 1 = 11$ times.

B.3. NON-I.I.D. DATA

Whereas in regular distributed learning environments, a characteristic of a typical federated learning environment is that data of the nodes is non-i.i.d. (not independent and identically distributed) [12, 52, 56]. In this thesis, we focus on a specific type of non-i.i.d. data, namely non-i.i.d. classes. When the classes are non-i.i.d., the number of samples available for each class differs between peers (but the data distribution of the samples available for a each class can still be i.i.d. between peers). This results in models that can be quite different between peers, making it hard for peers to distinguish between benign and malicious models. To make matters worse, even though McMahan et al. [7] originally showed that a trivial average of the parameter updates (FedAvg) achieves desirable accuracy in non-i.i.d. situations, this statement has been debunked by [7, 139, 140]: the model performs significantly worse than when it would have been trained by a single node on all data combined.

In Table B.2 we see that the number of GARS able to handle non-i.i.d. classes is very limited. In this section, we evaluate what techniques exist to learn non-i.i.d. classes when there are no Byzantine models present.

The challenge of building a single global model by combining multiple local models trained on different data distributions without reducing its accuracy is closely related to Lifelong Learning (often called Continual Learning in the deep learning community [142]). Continual Learning is concerned with preventing Catastrophic Forgetting or Catastrophic Inference, a phenomenon where a neural network completely forgets what it has learnt before when it is taught a new task. Instead, the network should be able to continuously acquire new knowledge, refine existing knowledge, and prevent new tasks from interfering with existing knowledge.

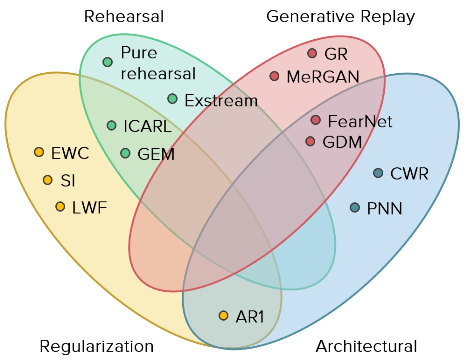


Figure B.1: Overview of existing CL methods [141]

Figure 4 is a Venn diagram created by Lesort and Lomonaco [141] categorizing the existing CL methods into four partially overlapping categories:

Architectural approaches seek to allocate additional neural nodes whenever they are required or freeze specific weights [143–145], but this requires the developer to know the number of tasks / samples per task a priori and leads to scalability issues for large neural networks. Two pioneers in this field are Lomonaco and Maltoni who first developed CWR [146], a dual-memory model that aims to replicate the hippocampus-cortex duality by selectively copying and resetting the output layer of the network. A year later, the authors extended CWR to CWR+ by implementing mean-shift and zero initialization for the output layer [147], and eventually to CWR* by replacing batch normalization with batch renormalization and weight constraining by learning rate modulation [148].

Regularization techniques minimize the extent to which the most important weights are overwritten by training on a new model. Elastic Weight Consolidation (EWC) [149], which was based on Learning without Forgetting (LwF) [150] is an influential regularization technique that extends the loss function with a quadratic penalty for the change in parameters important to previously learned tasks. The authors use the diagonal of the Fisher information matrix as a proxy for the importance of the parameters, which works well for learning permutations of the same task, but not for learning entirely new categories incrementally [151]. Several improvements have been made since such as [152–155].

Rehearsing old samples interleaved with new samples is also an effective way to prevent catastrophic forgetting. [140] concluded that globally sharing just 5% of the training data can result in 30% greater accuracy. These training samples can be selected randomly or carefully to be as representative of the coreset as possible. However, this approach increases the amount of memory needed to store all samples [156–159].

Generative replay is a variant of rehearsing old samples where a Generative Adversarial Network (GAN) is used to artificially generate samples that have a similar distribution as the past experiences. These samples are then intertwined with the new empirical training samples just like in rehearsal-based strategies [160–162].

The approaches discussed so far are generic multi-task learning techniques, but similar techniques have also been researched specifically for federated learning environments.

Chen et al. [163] present an example of a non-i.i.d. approach for federated learning, but the authors use clusters which do not work well on high-dimensional data (such as neural networks): the authors simply discard all parameters of the neural network except for the first 288 parameters in the first layer. The technique presented by Zhao et al. [140] is more effective and uses a rehearsal-based strategy: they assume that a small amount of i.i.d. data is available that can be shared across all peer nodes.

In specific situations where the loss function is convex and its conjugate dual is expressible, research has shown that dual coordinate ascent approaches such as Mocha en Cocoa can yield superior results [140, 164–166]. Mocha [166] handles non-i.i.d. datasets well and also tackles the challenge of fault tolerance, stragglers, and communication efficiency. The algorithm models the relation between the tasks by adding a loss term and subsequently uses a primal-dual formulation to solve the optimization problem. However, it assumes that all peers participate in each training round which makes Mocha inconvenient to use in a truly federated setting.

A particularly popular approach to use in federated learning systems seems to be Elastic Weight Consolidation ([167–171]) which, as explained earlier in this section, penalizes major changes of parameters that are important to previously learned tasks. It is somewhat surprising that more recent methods such as CWR(+/*), LWF, or AR1 have not been investigated yet because these methods perform significantly better in non-federated environments than EWC [172]. Our solution is based on CWR*[148], but we do want to mention that there is a small error in the original paper: the authors mention that their short-term memory implementation tw is modeled after the cortex region in the human brain and that their long-term memory implementation cw is modeled after the hippocampus. This is incorrect: the short-term memory tw is modeled after the prefrontal cortex, the long-term memory cw is modeled after the cerebral cortex, and the transfer mechanism between the short-term and long-term memory is (with a bit of imagination) modeled after the hippocampus [173]. We also think that working memory and consolidated memory are more accurate terms than short-term and long-term memory.

Another way of handling data heterogeneity is by combining the models of nodes with similar data distributions. For example, Bellet et al. [174] and Vanhaesebrouck et al. [175] presented fully decentralized federated learning systems where nodes learn their own personalized version of the model together with other nodes that have a similar data distribution. A major challenge is to determine which peers have a similar data distribution when the data distribution is private information, and how to determine if subtle variations in the data distributions are Byzantine or benign.

B.4. COMMUNICATION-EFFICIENCY

In DFL, each peer disseminates a copy of the current model at every training iteration to several other peers. However, modern neural networks may consist of millions of parameters [176, 177], sometimes requiring gigabytes of data to be transferred for each model exchange. Facilitating this amount of data quickly becomes infeasible when the number of devices and the frequency of model updates increase [178]. The following formula bounds the number of bits that are transmitted by every node during the training [179]:

$$b \in \mathcal{O}\left((H(\Delta\mathcal{W}) + \eta) * |\mathcal{W}| * N_{iter} * f\right) \tag{B.2}$$

where $H(\Delta\mathcal{W})$ is the entropy of the model's parameters, η is the difference between the minimum and the actual update size given a certain degree of entropy, $|\mathcal{W}|$ is the size of the model, N_{iter} is the total number of training iterations performed by the client, and f is the communication frequency. Each of these variables can be optimized to reduce the number of bits transmitted.

In the literature, the methods to decrease entropy of the model's parameters $H(\Delta\mathcal{W})$ are often categorized into quantization and sparsification techniques, although arguably a better categorization would be to distinguish between sketched updates and structured updates (of which quantization and sparsification techniques are an example respectively) [180]. Sketched updates refer to the compression of the full model update by performing structured random rotations, transmitting subsamples of the model which are then averaged to derive an unbiased estimate, or by using probabilistic quantization [181] where the gradients are quantized to low-precision values. The latter option has been researched extensively. Whereas SignSGD and its variants ([63, 182]) quantize the gradient of each parameter to a single bit, Feldman et al. [183] generalized the algorithm to a low-bit SGD version. Similar efforts were made by Alistarh et al. [184], Wen et al. [185], and Zhou et al. [186].

Structured updates refer to the restriction of the model updates to a pre-specified structure, i.e., low-rank structure or sparse matrix. A low-rank structure expresses each update as a product of two matrices, one of which is randomly generated and kept constant during the communication rounds, whereas the other is transmitted to the other nodes. Sparse matrix approaches are extremely popular and well-researched because the compression ratios are significantly higher than other methods such as quantization [184, 187]. Strom [188] noted that most parameters of a neural network are close to zero and therefore suggested to transmit only gradients greater than a predetermined constant threshold (a method later improved by [189]). Because this threshold is hard to determine, since it varies greatly depending on the layer and the architecture, many later works aimed to select gradients without using a fixed threshold. Dryden et al. [190] and Aji and Heafield [191] select a fixed proportion of the parameters to be transmitted, Chen et al. [192] adjust the compression rate dynamically based on local gradient activity, and Tao et al. [193] propose the eSGD algorithm that assigns weight values to the parameters based on an increase or decrease in the training loss and communicates only the parameters with the highest weights. Whereas all algorithms mentioned so far resulted in (a minor) loss of accuracy, Lin et al. [194] manage to achieve excellent compression results with no loss of accuracy by using momentum correction / factor masking, warm-up training, and gradient clipping. Instead of selectively communicating weights, Luping et al. [195] only transmit the model if the accuracy of the updated model is sufficiently higher than the former model. Tang et al. [196] introduce the ideas discussed so far in a decentralized environment.

There is also some research that focuses on decreasing the communication frequency, although this research is limited. Hu et al. [197] proposes ADSP that lets nodes transmit their model at strategically decided intervals, and Wang et al. [198] optimize the transmission frequency based on the resource budgets between all participating nodes.

Bristle significantly reduces the communication overhead by fixing the non-output layers of each peer and only transmitting the final output layer. This is an example of structured updating.

C
Implementation

In this section, we provide additional implementation details about the design of the entire system to increase the replicability of the study and improve the reader's understanding of the results. The full source code is publicly available on Github: the main repository[1] contains the source code of the Android application and the attacks / GARs, and the IPv8 repository[2] contains the source code of the coordinator program and the scripts used to generate the figures. All Kotlin code was written in Kotlin v1.4.32 and all Python code was written in Python 3.7 . The most notable libraries used are listed in Table C.1

Library	Version
org.deeplearning4j:deeplearning4j-core	1.0.0-beta7
org.nd4j:nd4j-native	1.0.0-beta7
org.bytedeco:opencv	4.4.0-1.5.4
org.bytedeco:leptonica	1.80.0-1.5.4
org.jetbrains.kotlinx:kotlinx-serialization-runtime	1.0-M1-1.4.0-rc

Table C.1: Most important libraries used

C.1. Environment

We use two separate ways to test Bristle's performance, namely by means of a centralized simulation on a single emulator and a decentralized simulation on up to 16 completely independent emulators. The experiments are run on an HPE DL385 Gen10 server equipped with 128 AMD EPYC 7452 CPUs, 512 GB of DDR4 memory, and Debian 10. The emulators are configured with 6GB of free disk space, 3GB of RAM, and Android 11 (API 30). This limit of 16 emulators is hardcoded in the Android Device Bridge (the software

[1] https://github.com/jverbraeken/trustchain-superapp/tree/master/fedml

[2] https://github.com/jverbraeken/kotlin-ipv8/tree/master/distributed-automation

used to communicate with Android devices), making it highly unpractical to run more emulators simultaneously. However, 16 emulators are enough to get a good idea of how the different GARs perform and therefore used for most experiments. For the local simulation, we run a single program that iteratively trains and combines up to 128 models to simulate a small-scale federated setting. Running an experiment in a local simulation is relatively slow because a single emulator can only use a single CPU core (so 16 emulators are able to use 16 CPU cores, significantly decreasing the experiment time). The results of the local simulation are similar to the results of the decentralized emulators when these emulators are configured to run at the same speed.

C.2. NETWORK PROTOCOL

The peers need to communicate with each other to be able to learn a model collaboratively. Since we envision Bristle to be used in decentralized systems, the peers need to be able to find each other and communicate with each other in a fault-tolerant and effective way. Therefore, we use IPv8 [199, 200], a well-established decentralized peer-to-peer (P2P) middleware stack used by i.a. the popular Tribler media sharing system [201, 202]. To enable peers to discover and communicate with other peers that use our FL technology, we create a new IPv8 community. Within this community, we define several new message types and create the corresponding message handlers to (a) communicate model updates, (b) send a heartbeat signal, (c) instruct the execution of a new experiment, (d) communicate the results of an evaluation, (e) signal that the experiment is finished, and (f) introduce a peer to multiple other peers simultaneously to decrease the time it takes for all peers to discover each other. Furthermore, we extend the communicate protocol of IPv8 with two significant performance enhancements to make the system more effective.

The first improvement is an extension to the Trivial File Transfer Protocol (TFTP). TFTP is the file transfer protocol used by IPv8 and is limited to transmitting only a single file at a time. Our extension enables parallel transmission of multiple files between the same two peers, implemented by assigning a unique file identifier to each file and prefixing each data packet with this identifier to keep track of all packets.

The second improvement is the implementation of a faster alternative to TFTP, namely uTP (micro-Transport Protocol). Our uTP implementation is the first one written in Kotlin. It significantly decreases the transmission time compared to TFTP and mitigates the poor latency and congestion control problems found in regular TCP implementations while providing reliable and ordered packet delivery. These advantages are realized by sending multiple packets simultaneously and slowing down the transmission rate when the network seems to get congested. Unfortunately, the uTP implementation is not entirely stable (as discussed in Section D.1), which is why we ended up using the modified TFTP version for the experiments.

C.3. EXPERIMENT ORCHESTRATION AND DEPLOYMENT

Running the experiment on each emulator is infeasible to do manually for a large number of emulators. Therefore, we created a separate coordinator program that orchestration the entire experiment pipeline. Based on the current operating system (Windows or

Linux) and the current state of the emulators (whether they exist, they are running, and root access is obtained) it executes several scripts to create, start, and initialize the emulators with the correct configuration. When the emulators are correctly initialized, we increase the network buffers significantly to prevent packets from being dropped due to insufficient space. By redirecting the ports, the emulators can communicate with each other through the host machine. The peers send periodically a heartbeat message to the coordinator to signal that they are still alive. When the coordinator misses multiple subsequent heartbeats from a peer, the peer is forcefully killed and restarted. The tasks are described in a dedicated JSON file, and the performance evaluations are written to a CSV file.

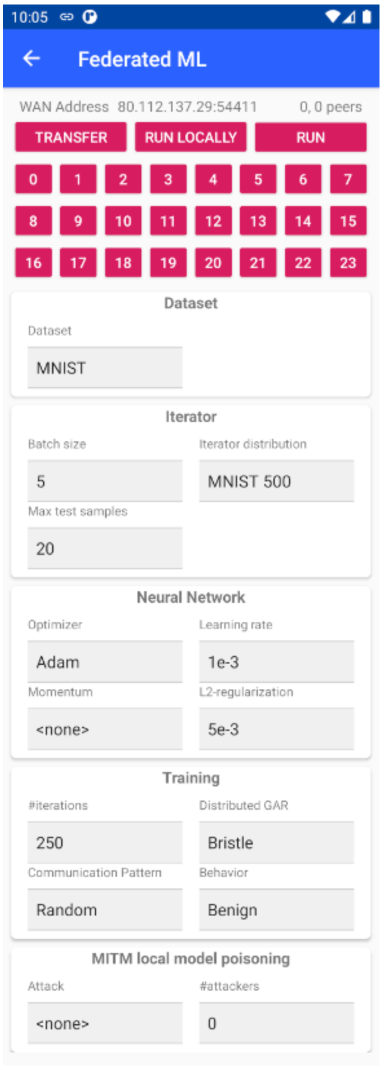
Subsequently, the evaluations are fed into the post-processing pipeline where they are collected, processed and eventually transformed into the desired figures.

C.4. INITIATION OF TESTS

There are three ways to run an experiment on the app that we designed, namely by using the user-interface, command-line arguments, or network requests.

C.4.1. USER-INTERFACE

The user interface was developed to give the user a convenient way to run a federated learning experiment and is illustrated in Figure C.1. The "transfer" button trains a transfer model on the dataset selected in the UI. The training is executed as configured by the user in the UI dropdown menus, uses all samples available in the dataset and stops after a pre-configured number of iterations. This number of iterations must be set manually to the number of iterations where the network is converged (i.e., the accuracy stops increasing) but is not yet overfitted. The trained model is saved on the emulator and can be copied to other emulators. The "run locally" button runs the experiment as configured in the dropdown menus as a regular (non-federated) ML experiment. This is useful to tune the network parameters when you have a new dataset. The experiments are run without transfer learning by default. The "run" button and all numeric buttons in the three rows below are used to simulate a subset of the experiments which is hard-coded by the developer for each button. The



WAN Address 80.112.137.29:54411 0, 0 peers

Dataset

Dataset: MNIST

Iterator

Batch size: 5
Iterator distribution: MNIST 500
Max test samples: 20

Neural Network

Optimizer: Adam
Learning rate: 1e-3
Momentum: <none>
L2-regularization: 5e-3

Training

#iterations: 250
Distributed GAR: Bristle
Communication Pattern: Random
Behavior: Benign

MITM local model poisoning

Attack: <none>
#attackers: 0

Figure C.1: A screenshot of the user interface

configuration of these simulations is defined in a JSON file instead of the UI, because a single button may run multiple simulations with different configurations. All experiments are run both without and with transfer learning. This means that the transfer model must be trained before the experiments are run and that Bristle (which depends on transfer learning) is skipped in the experiments without transfer learning.

C.4.2. COMMAND-LINE ARGUMENTS

Because clicking on the right buttons for each experiment can be tedious, another option is to launch the app with an array of command line arguments. These arguments have an equivalent function to the options available in the UI and allow the user to easily run a specific experiment with a specific configuration.

C.4.3. NETWORK REQUEST

The third way to run experiments is used to run them distributed over a set of independent emulators (see Section C.3). After the emulators are initialized, the coordinator program sends an experiment request including the exact training configuration to all peers simultaneously.

C.5. OPTIMIZATIONS

We optimized and re-implemented several methods to make them more efficient. The following list of optimizations is not exhaustive and may be extended in the future.

- Whereas the default dataset iterator implementation loads all training data from the external storage when a new experiment starts, we store the training data after it was loaded for the first time in a global memory pool that is accessed for subsequent runs.

- Whereas the default dataset iterator implementation requires the modification of the dataset to obtain a custom data distribution, we map, after the data was loaded for the first time, each class of the dataset to all corresponding samples. Subsequently, we can efficiently sample the right number of samples for each class for each experiment.

- Whenever the size of a data collection is known in advance, we use primitive arrays instead of lists or boxed primitive arrays to reduce the memory consumption and improve the speed with which elements at a certain index can be accessed.

- Whereas the Deeplearning4j method that converts an ML array to a primitive array performs checks on each element, we eliminated these checks to significantly decrease the computation time.

- For extremely large for-loops (namely, to construct the model poisoning Byzantine attacks), we increased the performance of the multiple invocations of the random access library function by segmenting this function into code that only needs to be run once and code that needs to be executed on every call.

- Using the profiler, we found out that the Kotlin code to calculate the minimum / maximum value of an array and to calculate a random number in a certain interval took a significant amount of time due to the large number of invocations. Therefore, we decided to micro-optimize these Kotlin functions by removing all checks and unused extension functionality, significantly increasing their performance.

D
Discussion

We believe that Bristle is an important step toward Byzantine-resilient and communication-efficient decentralized, federated learning with non-i.i.d. classes. Nevertheless, no software package is perfect and there are always certain aspects that can be improved. In this section, we aim to give the reader a large number of ideas for future research based on insights that we gained during the development of Bristle. Additionally, we list several of the biggest issues that we encountered during the research to give the reader a better understanding of the challenge to develop Bristle.

D.1. The road ahead

- In this thesis, we focus exclusively on cases where the classes are non-i.i.d. (i.e., the number of samples available for each class differs between peers) and where the data per class is i.i.d. (i.e., the data distribution of the samples for a given class is similar between all peers). Although the ability to handle non-i.i.d. classes is an important step forward, we did not investigate how Byzantine-resilience can be achieved when the data per class is non-i.i.d. One possibility is to add a Total Variation (TV) term that allows the integration of received models even when they perform slightly worse than the peer's own model on the peer's test dataset. This results in an inherent trade-off that can be mathematically formalized: let's define $\mathbb{P}$ as parameters trained on a different data distribution of which a peer has insufficient samples to estimate their benignity reliably. Then, a TV that increases the extent to which parameters $\mathbb{P}$ can be integrated results in a higher ability to learn non-i.i.d. data and in a lower ability to defend against Byzantine attacks.

- Bristle uses a distance-based prioritizer to reduce the number of models processed by the computationally expensive performance-based integrator. However, as discussed in the paper, calculating distances is unreliable when the data is highly non-i.i.d. Additionally, its computation time scales linearly with the number of parameters, which may become a problem when the number of parameters is very large. Future work may address these limitations by using an (additional) reputation-based filter that determines the peer reputation and uses it to discard models received from clearly Byzantine nodes.

- In the current implementation, we assume that the total number of classes is known a-priori. This limitation can be overcome by creating a global mapping between classes and CSPs (class-specific parameters; the parameters of the output layer responsible for the prediction of a particular class) and dynamically composing the output layer from all CSPs relevant for each peer.

- For this thesis, we used the MNIST dataset to evaluate Bristle. MNIST is among the most popular ML datasets used in the literature and especially popular for supervised classification problems. However, the dataset is relatively easy to learn which reduces the applicability of the results to complex real-world problems. A more challenging alternative is CIFAR-10, but the problem with this dataset is the opposite: it is incredibly difficult to learn, as illustrated by the poor performance in other studies[203–205]. A very popular and reasonably challenging dataset is ImageNet which contains a large number of classes with lots of heterogeneity between the samples. Unfortunately, we were unable to use such datasets for our experiments due to hardware limitations: the size of our hard disk is 500 GB, while the size of ImageNet is 150 GB (which must be stored on 16 distinct emulators, resulting in 150 x 16 = 2400 GB). Also the computation time is a considerable problem: the Android emulators are unable to use a GPU and are thus forced to execute the machine learning on the CPU instead. Unfortunately, CPUs are relatively slow to learn ML models. The time to run the relatively simple MNIST experiments is several days to run the current experiments and will be significantly longer for larger datasets.

- We chose to supply every peer with only 7 samples of each class: 2 to be used for training and 5 to be used for the performance-based integrator. This number of training samples is very low on purpose because it increases the challenge to achieve satisfactory accuracy and the benefit that can be gained from using the knowledge obtained by other peers. The number of samples used for the performance-based integrator entirely depends on the expected number of Byzantine attackers, the risk appetite of the user, and the degree of non-i.i.d.-ness between samples from the same class. Further research may develop concrete guidelines for the number of samples needed for the performance-based integrator.

- We evaluated Bristle by varying several parameters such as the attack type, transfer learning usage, number of Byzantine nodes, degree of non-i.i.d.-ness, and connection ratio. However, there are many more parameters that we can vary, which may yield surprising insights. In future work, we intend to evaluate the impact of momentum, batch normalization, use of different datasets, different number of hidden layers, different communication patterns, various modes of asynchrony, and highly imbalanced datasets on Bristle's performance.

- Bristle is based on the CL technique CWR*, which fixes the non-output layers and only updates the output layer. There are other methods like AR1 or EWC that are (in certain cases) more effective by allowing updates also the non-output layers. Unfortunately, it is hard to combine these methods with Bristle's mechanism to prevent Byzantine attacks: Bristle updates parameters based on their ability to predict a particular class correctly, but the only parameters associated with the prediction of just a single class are the parameters in the output layer. Future research may investigate this issue by using a multi-factor analysis that combines the difference in the accuracy of all classes with the weights of the neural connections between the output layer and the non-output layers.

- All current GARS (including Bristle) transmit the model after each training iteration. It might be possible to still achieve excellent results when the model is transmitted only after a number of iterations which can significantly reduce the communication costs.

- An important assumption made by Bristle is that for a given ML problem, there exists another large publicly available with roughly the same low-level features. Such a dataset is crucial for effective transfer learning. It would be fascinating to evaluate how "similar" a dataset should be to obtain satisfactory results.

- The uTP implementation is, as discussed in Section C.2, unstable and therefore not suitable for practical purposes. After a very large number of packets are sent/received, the packet delivery stalls for unknown reasons that are challenging to debug.

- Bristle was specifically developed for supervised classification problems. Future research may investigate how Bristle's architecture may be applied to other ML types, such as unsupervised ML or regression problems.

- Peers with a significant class overlap (i.e., between their familiar classes) are more likely to integrate each other's model than peers with very little class overlap. Therefore, the communication costs can be reduced by receiving models only from peers with a large class overlap. This class overlap can be measured while keeping the data distribution of each peer private by using private-set intersection cardinality (PSI-CA) methods. An efficient PSI-CA implementation was proposed by Shamir, Rivest, and Adleman [206], called SRA, and was later rediscovered (or not properly referenced) by Cristofaro et al. [207]. Although a peer cannot directly determine the exact classes that another peer owns by using PSI-CA, it is easy to obtain this information indirectly by simply submitting an exhaustive set of PSI-CA requests [208]. To avoid this, one may modify the first step of SRA by requiring the peers to submit at least a minimum number of classes (the higher the number, the better the confidentiality; when peers do not have enough classes, they can always add noise as padding) and enforce a limit on the number of PSI-CA requests accepted per peer and also on the number of PSI-CA request accepted in total per day (necessary because an attacker may create multiple distinct sybils that collaborate to determine the peer's classes). Unfortunately, given enough nodes and time, an attacker can eventually always determine which classes are owned by a particular peer due to the nature of the class-overlap problem, but these measures may make it significantly more difficult to do so for a large number of nodes.

- As discussed in Section B.1.2, targeted backdoor attacks are NP-hard to detect. Therefore, we did not focus on detecting this type of attack in this thesis. However, instead of detecting these attacks, it is possible to reduce the ability to execute them by pruning unused neurons (as mentioned in Section B.2.3). Future work might address the development of a Byzantine-resilient GAR that is highly effective against any attack rather than just one particular type.

D.2. BIGGEST ISSUES ENCOUNTERED

While working on my thesis, I encountered several major drawbacks that did cost me a significant amount of time. I want to highlight a few important ones to illustrate this:

- It was a challenging issue to make separate local emulators communicate to each other. First of all, it turned out that TFTP (the only network protocol available in IPv8) was unable to send and receive multiple files to/from the same peer simultaneously. After I rewrote it, it turned out that it was way too slow to transmit the entire model via localhost between all peers for every iteration. Therefore, I had to implement an entirely new network protocol, namely uTP. It is very time-consuming and intense to get a network protocol to run correctly, because there are many threads doing lots of things in parallel on multiple emulators and when the connection suddenly crashes after several minutes, it requires a lot of effort to debug where it is going wrong (for example, I had to replace HashMap by ConcurrentHashMap to prevent threading issues, but this caused deadlocks so I had to use proper mutexes and coroutines). Unfortunately, modern debugging software is still uncapable of breaking the program's execution at the moment of the crash and then go a few steps back, which complicates the debugging process significantly. It took more than a week to find out why packets sometimes got lost when transmitted over localhost: the local network buffers were too small.

- Another source of problems for network communication over localhost was the CPU scheduler of Linux. Originally, a peer sends a message to another peer and then writes to its memory that it had send the message. However, when the CPU scheduler decides to pause the peer just after it had sent a message to another peer, then lets the other peer respond, and then resumes the execution of the former peer, then the response of the other peer was received before the next line (writing to its memory that the peer had sent the message to the other peer), causing the program to crash in a way that was terrible to debug.

- An issue on which I could not find anything on the internet, and which took days to solve was that the debugger could not attach to the emulator. By pure luck I eventually discovered that it happened only when Android Studio and IntelliJ Idea are running simultaneously. I have reported this bug to IntelliJ.

- Another issue that I came across is that the performance profiler of Android Studio cannot stop profiling when a coroutine is being executed. It works correctly when I would change the coroutine to a thread. I have also reported this bug to IntelliJ.

- A very irritating limitation of developing for Android is that Google hard-coded a limit of 16 emulators to run simultaneously. The only way to run more emulators is to either run emulators inside other emulators, or to change the limit in the C++-code and recompile the emulator software. For the sake of time, I decided to just stick to 16 emulators.

- There are no proper deep-learning libraries available for Java and there are no Java ports available for proper deep-learning libraries in other languages. The best library currently available for Java is DeepLearning4j (DL4J), which is not being (seriously) maintained anymore[209]. Apart from the fact that fixing all dependencies took days (since all tutorials and documentation were outdated) and fixing issues such as incorrect internal rounding errors (adding and then subtracting gives in DL4J a different result than first subtracting and then adding) and working around hard-coded obsolete URLs inside the library was non-trivial, it also has a serious bug somewhere in its memory management, causing the library to crash when it trains multiple networks on different threads simultaneously. Since the source of this error is buried deep inside the C-layers, I decided to run the experiments sequentially.

REFERENCES

[1] H. Robbins and S. Monro, *A stochastic approximation method,* The annals of mathematical statistics , 400 (1951).

[2] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization,* arXiv preprint arXiv:1412.6980 (2014).

[3] M. C. Mukkamala and M. Hein, *Variants of rmsprop and adagrad with logarithmic regret bounds,* arXiv preprint arXiv:1706.05507 (2017).

[4] 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 *The Conference on Systems and Machine Learning (SysML), 2019.*

[5] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, *Scaling distributed machine learning with the parameter server,* in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 583–598.

[6] S. Zhang, A. E. Choromanska, and Y. LeCun, *Deep learning with elastic averaging sgd,* in *Advances in neural information processing systems*, pp. 685–693.

[7] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, *Communication-efficient learning of deep networks from decentralized data,* in *Artificial Intelligence and Statistics* (PMLR) pp. 1273–1282.

[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, and H. B. McMahan, *Towards federated learning at scale: System design,* arXiv preprint arXiv:1902.01046 (2019).

[9] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, *The loss surfaces of multilayer networks,* in *Artificial intelligence and statistics* (PMLR) pp. 192–204.

[10] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization,* arXiv preprint arXiv:1406.2572 (2014).

[11] I. J. Goodfellow, O. Vinyals, and A. M. Saxe, *Qualitatively characterizing neural network optimization problems,* arXiv preprint arXiv:1412.6544 (2014).

[12] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, *Byzantine-robust distributed learning: Towards optimal statistical rates,* arXiv preprint arXiv:1803.01498 (2018).

[13] B. Biggio, B. Nelson, and P. Laskov, *Poisoning attacks against support vector machines,* arXiv preprint arXiv:1206.6389 (2012).

[14] C. Fung, C. J. Yoon, and I. Beschastnikh, *Mitigating sybils in federated learning poisoning,* arXiv preprint arXiv:1808.04866 (2018).

[15] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, *Towards poisoning of deep learning algorithms with back-gradient optimization,* in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 27–38.

[16] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. D. Tygar, *Antidote: understanding and defending against poisoning of anomaly detectors,* in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pp. 1–14.

[17] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, *Data poisoning attacks against federated learning systems,* in *European Symposium on Research in Computer Security* (Springer) pp. 480–501.

[18] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, *Is feature selection secure against training data poisoning?* in *International Conference on Machine Learning*, pp. 1689–1698.

[19] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen, *Learning to detect malicious clients for robust federated learning,* arXiv preprint arXiv:2002.00211 (2020).

[20] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, *The hidden vulnerability of distributed learning in byzantium,* arXiv preprint arXiv:1802.07927 (2018).

[21] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, *Targeted backdoor attacks on deep learning systems using data poisoning,* arXiv preprint arXiv:1712.05526 (2017).

[22] T. Gu, B. Dolan-Gavitt, and S. Garg, *Badnets: Identifying vulnerabilities in the machine learning model supply chain,* arXiv preprint arXiv:1708.06733 (2017).

[23] P. W. Koh and P. Liang, *Understanding black-box predictions via influence functions,* arXiv preprint arXiv:1703.04730 (2017).

[24] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, *Data poisoning attacks on factorization-based collaborative filtering,* in *Advances in neural information processing systems*, pp. 1885–1893.

[25] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, *Poison frogs! targeted clean-label poisoning attacks on neural networks,* in *Advances in Neural Information Processing Systems*, pp. 6103–6113.

[26] 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*, pp. 508–519.

[27] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras, *When does machine learning FAIL? generalized transferability for evasion and poisoning attacks,* in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1299–1316.

[28] C. Xie, K. Huang, P.-Y. Chen, and B. Li, *Dba: Distributed backdoor attacks against federated learning,* in *International Conference on Learning Representations.*

[29] M. Zou, Y. Shi, C. Wang, F. Li, W. Song, and Y. Wang, *Potrojan: powerful neural-level trojan designs in deep learning models,* arXiv preprint arXiv:1802.03043 (2018).

[30] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, *How to backdoor federated learning,* in *International Conference on Artificial Intelligence and Statistics* (PMLR) pp. 2938–2948.

[31] G. Baruch, M. Baruch, and Y. Goldberg, *A little is enough: Circumventing defenses for distributed learning,* in *Advances in Neural Information Processing Systems*, pp. 8635–8645.

[32] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, *Model poisoning attacks in federated learning,* in *In Workshop on Security in Machine Learning (SecML), collocated with the 32nd Conference on Neural Information Processing Systems (NeurIPS'18).*

[33] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, *Analyzing federated learning through an adversarial lens,* in *International Conference on Machine Learning* (PMLR) pp. 634–643.

[34] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, *Trojaning attack on neural networks,* (2017).

[35] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, *Can you really backdoor federated learning?* arXiv preprint arXiv:1911.07963 (2019).

[36] L. Muñoz-González, K. T. Co, and E. C. Lupu, *Byzantine-robust federated machine learning through adaptive model averaging,* arXiv preprint arXiv:1909.05125 (2019).

[37] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, *Deep learning with differential privacy,* in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318.

[38] C. Dwork, *Differential privacy: A survey of results,* in *International conference on theory and applications of models of computation* (Springer) pp. 1–19.

[39] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, *Federated learning with differential privacy: Algorithms and performance analysis,* IEEE Transactions on Information Forensics and Security **15**, 3454 (2020).

[40] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, and R. Cummings, *Advances and open problems in federated learning,* arXiv preprint arXiv:1912.04977 (2019).

[41] M. Fang, X. Cao, J. Jia, and N. Gong, *Local model poisoning attacks to byzantine-robust federated learning,* in *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1605–1622.

[42] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, *Manipulating machine learning: Poisoning attacks and countermeasures for regression learning,* in *2018 IEEE Symposium on Security and Privacy (SP)* (IEEE) pp. 19–35.

[43] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee, and D. Papailiopoulos, *Attack of the tails: Yes, you really can backdoor federated learning,* arXiv preprint arXiv:2007.05084 (2020).

[44] E.-M. El-Mhamdi and R. Guerraoui, *Fast and secure distributed learning in high dimension,* arXiv preprint arXiv:1905.04374 (2019).

[45] S. Haykin, *Neural Networks and Learning Machines, 3/E* (Pearson Education India, 2010).

[46] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, *On large-batch training for deep learning: Generalization gap and sharp minima,* arXiv preprint arXiv:1609.04836 (2016).

[47] R. Kleinberg, Y. Li, and Y. Yuan, *An alternative view: When does sgd escape local minima?* arXiv preprint arXiv:1802.06175 (2018).

[48] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, *Adding gradient noise improves learning for very deep networks,* arXiv preprint arXiv:1511.06807 (2015).

[49] L. Bottou, *Online learning and stochastic approximations,* On-line learning in neural networks **17**, 142 (1998).

[50] C. Xie, O. Koyejo, and I. Gupta, *Generalized byzantine-tolerant sgd,* arXiv preprint arXiv:1802.10116 (2018).

[51] S. Guo, T. Zhang, X. Xie, L. Ma, T. Xiang, and Y. Liu, *Towards byzantine-resilient learning in decentralized systems,* arXiv preprint arXiv:2002.08569 (2020).

[52] P. Blanchard, R. Guerraoui, and J. Stainer, *Machine learning with adversaries: Byzantine tolerant gradient descent,* in *Advances in Neural Information Processing Systems,* pp. 119–129.

[53] Z. Yang, A. Gang, and W. U. Bajwa, *Adversary-resilient inference and machine learning: From distributed to decentralized,* stat **1050**, 23 (2019).

[54] L. Su and N. H. Vaidya, *Fault-tolerant distributed optimization (part iv): Constrained optimization with arbitrary directed networks,* arXiv preprint arXiv:1511.01821 (2015).

[55] S. Sundaram and B. Gharesifard, *Distributed optimization under adversarial nodes,* IEEE Transactions on Automatic Control **64**, 1063 (2018).

[56] Y. Chen, L. Su, and J. Xu, *Distributed statistical machine learning in adversarial settings: Byzantine gradient descent,* Proceedings of the ACM on Measurement and Analysis of Computing Systems **1**, 1 (2017).

[57] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, *Draco: Byzantine-resilient distributed training via redundant gradients,* arXiv preprint arXiv:1803.09877 (2018).

[58] N. Alon, Y. Matias, and M. Szegedy, *The space complexity of approximating the frequency moments,* Journal of Computer and system sciences **58**, 137 (1999).

[59] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani, *Random generation of combinatorial structures from a uniform distribution,* Theoretical computer science **43**, 169 (1986).

[60] M. Lerasle and R. I. Oliveira, *Robust empirical mean estimators,* arXiv preprint arXiv:1112.3914 (2011).

[61] S. Minsker, *Geometric median and robust estimation in banach spaces,* Bernoulli **21**, 2308 (2015).

[62] S. Minsker, *Distributed statistical estimation and rates of convergence in normal approximation,* Electronic Journal of Statistics **13**, 5213 (2019).

[63] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, *signsgd: Compressed optimisation for non-convex problems,* arXiv preprint arXiv:1802.04434 (2018).

[64] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, *signsgd with majority vote is communication efficient and fault tolerant,* arXiv preprint arXiv:1810.05291 (2018).

[65] X. Chen, T. Chen, H. Sun, Z. S. Wu, and M. Hong, *Distributed training with heterogeneous data: Bridging median-and mean-based algorithms,* arXiv preprint arXiv:1906.01736 (2019).

[66] J.-y. Sohn, D.-J. Han, B. Choi, and J. Moon, *Election coding for distributed learning: Protecting signsgd against byzantine attacks,* arXiv preprint arXiv:1910.06093 (2019).

[67] L. Li, W. Xu, T. Chen, G. B. Giannakis, and Q. Ling, *Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 1544–1551.

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

[69] Z. Yang and W. U. Bajwa, *Bridge: Byzantine-resilient decentralized gradient descent,* arXiv preprint arXiv:1908.08098 (2019).

[70] J. Peng and Q. Ling, *Byzantine-robust decentralized stochastic optimization,* in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE) pp. 5935–5939.

[71] L. He, S. P. Karimireddy, and M. Jaggi, *Byzantine-robust learning on heterogeneous datasets via resampling,* arXiv preprint arXiv:2006.09365 (2020).

[72] N. Gupta, S. Liu, and N. H. Vaidya, *Byzantine fault-tolerant distributed machine learning using stochastic gradient descent (sgd) and norm-based comparative gradient elimination (cge),* arXiv preprint arXiv:2008.04699 (2020).

[73] D. Cao, S. Chang, Z. Lin, G. Liu, and D. Sun, *Understanding distributed poisoning attack in federated learning,* in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)* (IEEE) pp. 233–239.

[74] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar, *The security of machine learning,* Machine Learning **81**, 121 (2010).

[75] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis, *Casting out demons: Sanitizing training data for anomaly sensors,* in *2008 IEEE Symposium on Security and Privacy (sp 2008)* (IEEE) pp. 81–95.

[76] R. Jin, X. He, and H. Dai, *Distributed byzantine tolerant stochastic gradient descent in the era of big data,* in *ICC 2019-2019 IEEE International Conference on Communications (ICC)* (IEEE) pp. 1–6.

[77] B. Tran, J. Li, and A. Madry, *Spectral signatures in backdoor attacks,* in *Advances in Neural Information Processing Systems,* pp. 8000–8010.

[78] L. Zhao, S. Hu, Q. Wang, J. Jiang, S. Chao, X. Luo, and P. Hu, *Shielding collaborative learning: Mitigating poisoning attacks through client-side detection,* IEEE Transactions on Dependable and Secure Computing (2020).

[79] X. Cao and L. Lai, *Distributed gradient descent algorithm robust to an arbitrary number of byzantine attackers,* IEEE Transactions on Signal Processing **67**, 5850 (2019).

[80] J. Ji, X. Chen, Q. Wang, L. Yu, and P. Li, *Learning to learn gradient aggregation by gradient descent,* in *IJCAI,* pp. 2614–2620.

[81] J. Regatti and A. Gupta, *Befriending the byzantines through reputation scores,* arXiv preprint arXiv:2006.13421 (2020).

[82] C. Xie, S. Koyejo, and I. Gupta, *Zeno++: Robust fully asynchronous sgd,* arXiv preprint arXiv:1903.07020 (2019).

[83] C. Xie, S. Koyejo, and I. Gupta, *Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance,* in *International Conference on Machine Learning* (PMLR) pp. 6893–6901.

[84] Y. Zhao, J. Chen, J. Zhang, D. Wu, J. Teng, and S. Yu, *Pdgan: A novel poisoning defense method in federated learning using generative adversarial network,* in *International Conference on Algorithms and Architectures for Parallel Processing* (Springer) pp. 595–609.

[85] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, *Beyond inferring class representatives: User-level privacy leakage from federated learning,* in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications* (IEEE) pp. 2512–2520.

[86] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, *A survey on security and privacy of federated learning,* Future Generation Computer Systems (2020).

[87] Y. Jiang, S. Wang, B. J. Ko, W.-H. Lee, and L. Tassiulas, *Model pruning enables efficient federated learning on edge devices,* arXiv preprint arXiv:1909.12326 (2019).

[88] K. Liu, B. Dolan-Gavitt, and S. Garg, *Fine-pruning: Defending against backdooring attacks on deep neural networks,* in *International Symposium on Research in Attacks, Intrusions, and Defenses* (Springer) pp. 273–294.

[89] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, *Neural cleanse: Identifying and mitigating backdoor attacks in neural networks,* in *2019 IEEE Symposium on Security and Privacy (SP)* (IEEE) pp. 707–723.

[90] P. W. Koh, J. Steinhardt, and P. Liang, *Stronger data poisoning attacks break data sanitization defenses,* arXiv preprint arXiv:1811.00741 (2018).

[91] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, *Detecting backdoor attacks on deep neural networks by activation clustering,* arXiv preprint arXiv:1811.03728 (2018).

[92] E. Chou, F. Tramèr, G. Pellegrino, and D. Boneh, *Sentinet: Detecting physical attacks against deep learning systems,* arXiv preprint arXiv:1812.00292 (2018).

[93] I. Diakonikolas, G. Kamath, D. Kane, J. Li, J. Steinhardt, and A. Stewart, *Sever: A robust meta-algorithm for stochastic optimization,* in *International Conference on Machine Learning,* pp. 1596–1606.

[94] M. Qiao and G. Valiant, *Learning discrete distributions from untrusted batches,* arXiv preprint arXiv:1711.08113 (2017).

[95] Y. Shen and S. Sanghavi, *Learning with bad training data via iterative trimmed loss minimization,* in *International Conference on Machine Learning* (PMLR) pp. 5739–5748.

[96] J. Steinhardt, P. W. W. Koh, and P. S. Liang, *Certified defenses for data poisoning attacks,* in *Advances in neural information processing systems,* pp. 3517–3529.

[97] S. Azulay, L. Raz, A. Globerson, T. Koren, and Y. Afek, *Holdout sgd: Byzantine tolerant federated learning,* arXiv preprint arXiv:2008.04612 (2020).

[98] X. Bao, C. Su, Y. Xiong, W. Huang, and Y. Hu, *Flchain: A blockchain for auditable federated learning with trust and incentive,* in *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)* (IEEE) pp. 151–159.

[99] X. Chen, J. Ji, C. Luo, W. Liao, and P. Li, *When machine learning meets blockchain: A decentralized, privacy-preserving and secure design,* in *2018 IEEE International Conference on Big Data (Big Data)* (IEEE) pp. 1178–1187.

[100] H. Kim, J. Park, M. Bennis, and S.-L. Kim, *Blockchained on-device federated learning,* IEEE Communications Letters **24**, 1279 (2019).

[101] H. Kim, S.-H. Kim, J. Y. Hwang, and C. Seo, *Efficient privacy-preserving machine learning for blockchain network,* IEEE Access **7**, 136481 (2019).

[102] Y. J. Kim and C. S. Hong, *Blockchain-based node-aware dynamic weighting methods for improving federated learning performance,* in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)* (IEEE) pp. 1–4.

[103] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, *Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles,* IEEE Transactions on Vehicular Technology **69**, 4298 (2020).

[104] U. Majeed and C. S. Hong, *Flchain: Federated learning via mec-enabled blockchain network,* in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)* (IEEE) pp. 1–4.

[105] K. Salah, M. H. U. Rehman, N. Nizamuddin, and A. Al-Fuqaha, *Blockchain for ai: Review and open research challenges,* IEEE Access **7**, 10127 (2019).

[106] R. Schmid, B. Pfitzner, J. Beilharz, B. Arnrich, and A. Polze, *Tangle ledger for decentralized learning,* in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (IEEE) pp. 852–859.

[107] M. Shayan, C. Fung, C. J. Yoon, and I. Beschastnikh, *Biscotti: A ledger for private and secure peer-to-peer machine learning,* arXiv preprint arXiv:1811.09904 (2018).

[108] K. TOYODA, P. T. MATHIOPOULOS, and A. N. ZHANG, *Novel blockchain-based incentive-aware federated learning platform with mechanism design,* .

[109] K. Toyoda and A. N. Zhang, *Mechanism design for an incentive-aware blockchain-enabled federated learning platform,* in *2019 IEEE International Conference on Big Data (Big Data)* (IEEE) pp. 395–403.

[110] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang, and W. Luo, *Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive,* IEEE Transactions on Dependable and Secure Computing (2019).

[111] Y. Zhao, J. Zhao, L. Jiang, R. Tan, and D. Niyato, *Mobile edge computing, blockchain and reputation-based crowdsourcing iot federated learning: A secure, decentralized and privacy-preserving system,* arXiv preprint arXiv:1906.10893 (2019).

[112] S. Zhou, H. Huang, W. Chen, P. Zhou, Z. Zheng, and S. Guo, *Pirate: A blockchain-based secure framework of distributed machine learning in 5g networks,* IEEE Network (2020).

[113] J. Kang, Z. Xiong, D. Niyato, H. Yu, Y.-C. Liang, and D. I. Kim, *Incentive design for efficient federated learning in mobile networks: A contract theory approach,* in *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)* (IEEE) pp. 1–5.

[114] J. Kang, Z. Xiong, D. Niyato, S. Xie, and J. Zhang, *Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory,* IEEE Internet of Things Journal **6**, 10700 (2019).

[115] Y. Zhao, J. Zhao, L. Jiang, R. Tan, D. Niyato, Z. Li, L. Lyu, and Y. Liu, *Privacy-preserving blockchain-based federated learning for iot devices,* IEEE Internet of Things Journal (2020).

[116] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor, *Chained anomaly detection models for federated learning: An intrusion detection case study,* Applied Sciences **8**, 2663 (2018).

[117] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, *Algorand: Scaling byzantine agreements for cryptocurrencies,* in *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 51–68.

[118] M. Cong, H. Yu, X. Weng, J. Qu, Y. Liu, and S. M. Yiu, *A vcg-based fair incentive mechanism for federated learning,* arXiv preprint arXiv:2008.06680 (2020).

[119] N. Ding, Z. Fang, and J. Huang, *Incentive mechanism design for federated learning with multi-dimensional private information,* in *2020 18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)* (IEEE) pp. 1–8.

[120] R. Hu and Y. Gong, *Trading data for learning: Incentive mechanism for on-device federated learning,* arXiv preprint arXiv:2009.05604 (2020).

[121] L. U. Khan, N. H. Tran, S. R. Pandey, W. Saad, Z. Han, M. N. Nguyen, and C. S. Hong, *Federated learning for edge networks: Resource optimization and incentive mechanism,* arXiv preprint arXiv:1911.05642 (2019).

[122] T. H. T. Le, N. H. Tran, Y. K. Tun, M. N. Nguyen, S. R. Pandey, Z. Han, and C. S. Hong, *An incentive mechanism for federated learning in wireless cellular network: An auction approach,* arXiv preprint arXiv:2009.10269 (2020).

[123] W. Y. B. Lim, Z. Xiong, J. Kang, D. Niyato, Y. Zhang, C. Leung, and C. Miao, *An incentive scheme for federated learning in the sky,* in *Proceedings of the 2nd ACM MobiCom Workshop on Drone Assisted Wireless Communications for 5G and Beyond*, pp. 55–60.

[124] W. Y. B. Lim, Z. Xiong, C. Miao, D. Niyato, Q. Yang, C. Leung, and H. V. Poor, *Hierarchical incentive mechanism design for federated machine learning in mobile networks,* IEEE Internet of Things Journal (2020).

[125] K. L. Ng, Z. Chen, H. Y. Zelei Liu, Y. Liu, and Q. Yang, *A multi-player game for studying federated learning incentive schemes,* .

[126] S. R. Pandey, S. Suhail, Y. K. Tun, M. Alsenwi, and C. S. Hong, *An incentive design to perform federated learning,* .

[127] H. Yu, Z. Liu, Y. Liu, T. Chen, M. Cong, X. Weng, D. Niyato, and Q. Yang, *A fairness-aware incentive scheme for federated learning,* in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pp. 393–399.

[128] H. Yu, Z. Liu, Y. Liu, T. Chen, M. Cong, X. Weng, D. Niyato, and Q. Yang, *A sustainable incentive scheme for federated learning,* IEEE Intelligent Systems (2020).

[129] R. Zeng, S. Zhang, J. Wang, and X. Chu, *Fmore: An incentive scheme of multi-dimensional auction for federated learning in mec,* arXiv preprint arXiv:2002.09699 (2020).

[130] Y. Zhan, P. Li, Z. Qu, D. Zeng, and S. Guo, *A learning-based incentive mechanism for federated learning,* IEEE Internet of Things Journal (2020).

[131] S. Feng, D. Niyato, P. Wang, D. I. Kim, and Y.-C. Liang, *Joint service pricing and co-operative relay communication for federated learning,* in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (IEEE) pp. 815–820.

[132] Y. Sarikaya and O. Ercetin, *Motivating workers in federated learning: A stackelberg game perspective,* IEEE Networking Letters **2**, 23 (2019).

[133] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, *Practical secure aggregation for federated learning on user-held data,* arXiv preprint arXiv:1611.04482 (2016).

[134] Y. Chen, F. Luo, T. Li, T. Xiang, Z. Liu, and J. Li, *A training-integrity privacy-preserving federated learning scheme with trusted execution environment,* Information Sciences **522**, 69 (2020).

[135] F. Mo and H. Haddadi, *Efficient and private federated learning using tee,* in *EuroSys.*

[136] M. Sabt, M. Achemlal, and A. Bouabdallah, *Trusted execution environment: what it is, and what it is not,* in *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1 (IEEE) pp. 57–64.

[137] D. Data, L. Song, and S. Diggavi, *Data encoding for byzantine-resilient distributed optimization,* arXiv preprint arXiv:1907.02664 (2019).

[138] 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*, pp. 10320–10330.

[139] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, *Adaptive federated learning in resource constrained edge computing systems,* IEEE Journal on Selected Areas in Communications **37**, 1205 (2019).

[140] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, *Federated learning with non-iid data,* arXiv preprint arXiv:1806.00582 (2018).

[141] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, *Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges,* Information fusion **58**, 52 (2020).

[142] Y. Zhang and Q. Yang, *A survey on multi-task learning,* arXiv preprint arXiv:1707.08114 (2017).

[143] J. A. Hertz, *Introduction to the theory of neural computation* (CRC Press, 2018).

[144] G. I. Parisi, J. Tani, C. Weber, and S. Wermter, *Lifelong learning of human actions with deep neural network self-organization,* Neural Networks **96**, 137 (2017).

[145] G. I. Parisi, J. Tani, C. Weber, and S. Wermter, *Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization,* Frontiers in neurorobotics **12**, 78 (2018).

[146] V. Lomonaco and D. Maltoni, *Core50: a new dataset and benchmark for continuous object recognition,* in *Conference on Robot Learning* (PMLR) pp. 17–26.

[147] D. Maltoni and V. Lomonaco, *Continuous learning in single-incremental-task scenarios,* Neural Networks **116**, 56 (2019).

[148] V. Lomonaco, D. Maltoni, and L. Pellegrini, *Rehearsal-free continual learning over small non-iid batches,* arXiv preprint arXiv:1907.03799 (2019).

[149] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, *Overcoming catastrophic forgetting in neural networks,* Proceedings of the national academy of sciences **114**, 3521 (2017).

[150] Z. Li and D. Hoiem, *Learning without forgetting,* IEEE transactions on pattern analysis and machine intelligence **40**, 2935 (2017).

[151] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, *Measuring catastrophic forgetting in neural networks,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[152] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, *Overcoming catastrophic forgetting by incremental moment matching,* arXiv preprint arXiv:1703.08475 (2017).

[153] X. Liu, M. Masana, L. Herranz, J. Van de Weijer, A. M. Lopez, and A. D. Bagdanov, *Rotate your networks: Better weight consolidation and less catastrophic forgetting,* in *2018 24th International Conference on Pattern Recognition (ICPR)* (IEEE) pp. 2262–2268.

[154] H. Ritter, A. Botev, and D. Barber, *Online structured laplace approximations for overcoming catastrophic forgetting,* arXiv preprint arXiv:1805.07810 (2018).

[155] F. Zenke, B. Poole, and S. Ganguli, *Continual learning through synaptic intelligence,* in *International Conference on Machine Learning* (PMLR) pp. 3987–3995.

[156] A. Gepperth and C. Karaoguz, *A bio-inspired incremental learning architecture for applied perceptual problems,* Cognitive Computation **8**, 924 (2016).

[157] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, *icarl: Incremental classifier and representation learning,* in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010.

[158] A. Robins, *Catastrophic forgetting in neural networks: the role of rehearsal mechanisms,* in *Proceedings 1993 The First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems* (IEEE) pp. 65–68.

[159] A. Robins, *Catastrophic forgetting, rehearsal and pseudorehearsal,* Connection Science **7**, 123 (1995).

[160] D. C. Mocanu, M. T. Vega, E. Eaton, P. Stone, and A. Liotta, *Online contrastive divergence with generative replay: Experience replay without storing data,* arXiv preprint arXiv:1610.05555 (2016).

[161] H. Shin, J. K. Lee, J. Kim, and J. Kim, *Continual learning with deep generative replay,* arXiv preprint arXiv:1705.08690 (2017).

[162] G. M. Van de Ven and A. S. Tolias, *Generative replay with feedback connections as a general strategy for continual learning,* arXiv preprint arXiv:1809.10635 (2018).

[163] Z. Chen, P. Tian, W. Liao, and W. Yu, *Zero knowledge clustering based adversarial mitigation in heterogeneous federated learning,* IEEE Transactions on Network Science and Engineering (2020).

[164] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan, *Communication-efficient distributed dual coordinate ascent,* Advances in neural information processing systems **27**, 3068 (2014).

[165] C. Ma, J. Konečný, M. Jaggi, V. Smith, M. I. Jordan, P. Richtárik, and M. Takáč, *Distributed optimization with arbitrary local solvers,* Optimization Methods and Software **32**, 813 (2017).

[166] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, *Federated multi-task learning,* in *Advances in Neural Information Processing Systems*, pp. 4424–4434.

[167] C. Gonzalez, G. Sakas, and A. Mukhopadhyay, *What is wrong with continual learning in medical image segmentation?* arXiv preprint arXiv:2010.11008 (2020).

[168] K. Kopparapu and E. Lin, *Fedfmc: Sequential efficient federated learning on non-iid data,* arXiv preprint arXiv:2006.10937 (2020).

[169] S. Kumar, S. Dutta, S. Chatturvedi, and M. Bhatia, *Strategies for enhancing training and privacy in blockchain enabled federated learning,* in *2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM)* (IEEE) pp. 333–340.

[170] C. X. Ling and T. Bohn, *A conceptual framework for lifelong learning,* arXiv preprint arXiv:1911.09704 (2019).

[171] X. Yao and L. Sun, *Continual local training for better initialization of federated models,* in *2020 IEEE International Conference on Image Processing (ICIP)* (IEEE) pp. 1736–1740.

[172] V. Lomonaco, *Continual learning with deep architectures,* (2019).

[173] A. R. Preston and H. Eichenbaum, *Interplay of hippocampus and prefrontal cortex in memory,* Current Biology **23**, R764 (2013).

[174] A. Bellet, R. Guerraoui, M. Taziki, and M. Tommasi, *Personalized and private peer-to-peer machine learning,* in *International Conference on Artificial Intelligence and Statistics* (PMLR) pp. 473–481.

[175] P. Vanhaesebrouck, A. Bellet, and M. Tommasi, *Decentralized collaborative learning of personalized models over networks,* in *Artificial Intelligence and Statistics* (PMLR) pp. 509–517.

[176] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 770–778.

[177] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017) pp. 4700–4708.

[178] F. Sattler, T. Wiegand, and W. Samek, *Trends and advancements in deep neural network communication,* arXiv preprint arXiv:2003.03320 (2020).

[179] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, *Robust and communication-efficient federated learning from non-iid data,* IEEE transactions on neural networks and learning systems **31**, 3400 (2019).

[180] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, *Federated learning: Strategies for improving communication efficiency,* arXiv preprint arXiv:1610.05492 (2016).

[181] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,* arXiv preprint arXiv:1510.00149 (2015).

[182] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, *1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns,* in *Fifteenth Annual Conference of the International Speech Communication Association.*

[183] C. De Sa, M. Feldman, C. Ré, and K. Olukotun, *Understanding and optimizing asynchronous low-precision stochastic gradient descent,* in *Proceedings of the 44th Annual International Symposium on Computer Architecture,* pp. 561–574.

[184] D. Alistarh, J. Li, R. Tomioka, and M. Vojnovic, *Qsgd: Randomized quantization for communication-optimal stochastic gradient descent,* arXiv preprint arXiv:1610.02132 **1** (2016).

[185] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, *Terngrad: Ternary gradients to reduce communication in distributed deep learning,* arXiv preprint arXiv:1705.07878 (2017).

[186] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, *Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,* arXiv preprint arXiv:1606.06160 (2016).

[187] J. Konečný and P. Richtárik, *Randomized distributed mean estimation: Accuracy vs. communication,* Frontiers in Applied Mathematics and Statistics **4**, 62 (2018).

[188] N. Strom, *Scalable distributed dnn training using commodity gpu cloud computing,* in *Sixteenth Annual Conference of the International Speech Communication Association.*

[189] Y. Tsuzuku, H. Imachi, and T. Akiba, *Variance-based gradient compression for efficient distributed deep learning,* arXiv preprint arXiv:1802.06058 (2018).

[190] N. Dryden, T. Moon, S. A. Jacobs, and B. Van Essen, *Communication quantization for data-parallel training of deep neural networks,* in *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)* (IEEE) pp. 1–8.

[191] A. F. Aji and K. Heafield, *Sparse communication for distributed gradient descent,* arXiv preprint arXiv:1704.05021 (2017).

[192] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, *Adacomp: Adaptive residual gradient compression for data-parallel distributed training,* in *Proceedings of the AAAI Conference on Artificial Intelligence,* Vol. 32.

[193] Z. Tao and Q. Li, *esgd: Communication efficient distributed deep learning on the edge,* in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18).*

[194] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, *Deep gradient compression: Reducing the communication bandwidth for distributed training,* arXiv preprint arXiv:1712.01887 (2017).

[195] W. Luping, W. Wei, and L. Bo, *Cmfl: Mitigating communication overhead for federated learning,* in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)* (IEEE) pp. 954–964.

[196] H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu, *Communication compression for decentralized training,* in *Advances in Neural Information Processing Systems,* pp. 7652–7662.

[197] H. Hu, D. Wang, and C. Wu, *Distributed machine learning through heterogeneous edge systems,* in *Proceedings of the AAAI Conference on Artificial Intelligence,* Vol. 34, pp. 7179–7186.

[198] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, *When edge meets learning: Adaptive control for resource-constrained distributed machine learning,* in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications* (IEEE) pp. 63–71.

[199] Q. Stokkink, D. Epema, and J. Pouwelse, *A truly self-sovereign identity system,* arXiv preprint arXiv:2007.00415 (2020).

[200] Q. Stokkink and J. Pouwelse, *Deployment of a blockchain-based self-sovereign identity,* in *2018 IEEE international conference on Internet of Things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)* (IEEE) pp. 1336–1342.

[201] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. Epema, M. Reinders, M. R. Van Steen, and H. J. Sips, *Tribler: a social-based peer-to-peer system,* Concurrency and computation: Practice and experience **20**, 127 (2008).

[202] N. Zeilemaker, M. Capotă, A. Bakker, and J. Pouwelse, *Tribler: P2p media search and sharing,* in *Proceedings of the 19th ACM international conference on Multimedia,* pp. 739–742.

[203] G. Damaskinos, R. Guerraoui, A.-M. Kermarrec, V. Nitu, R. Patra, and F. Taiani, *Fleet: Online federated learning via staleness awareness and performance prediction,* in *Proceedings of the 21st International Middleware Conference* (2020) pp. 163–177.

[204] M. Yurochkin, M. Agarwal, S. Ghosh, K. Greenewald, N. Hoang, and Y. Khazaeni, *Bayesian nonparametric federated learning of neural networks,* in *International Conference on Machine Learning* (PMLR, 2019) pp. 7252–7261.

[205] W. Wan, J. Lu, S. Hu, L. Y. Zhang, and X. Pei, *Shielding federated learning: A new attack approach and its defense,* in *2021 IEEE Wireless Communications and Networking Conference (WCNC)* (IEEE, 2021) pp. 1–7.

[206] A. Shamir, R. L. Rivest, and L. M. Adleman, *Mental poker,* in *The mathematical gardner* (Springer, 1981) pp. 37–43.

[207] E. De Cristofaro, P. Gasti, and G. Tsudik, *Fast and private computation of cardinality of set intersection and union,* in *International Conference on Cryptology and Network Security* (Springer) pp. 218–231.

[208] K. Holzapfel, M. Karl, L. Lotz, G. Carle, C. Djeffal, C. Fruck, C. Haack, D. Heckmann, P. H. Kindt, and M. Köppl, *Digital contact tracing service: An improved decentralized design for privacy and effectiveness,* arXiv preprint arXiv:2006.16960 (2020).

[209] *What happened to Deeplearning4j?* https://www.reddit.com/r/deeplearning4j/comments/ie7b2o/what_happened_to_deeplearnign4j/, [Online; accessed 05-Jan-2021].