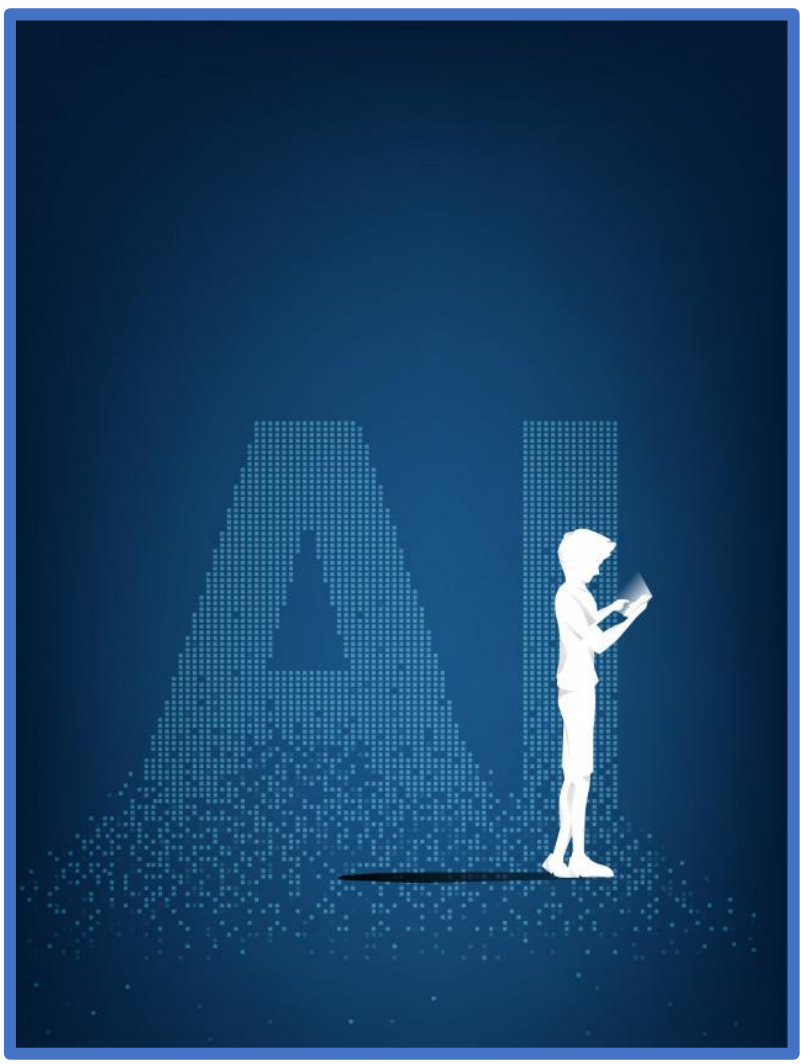
Joost Verbraeken
Supervised by Dr. Pouwelse and Dr. de Vos

# Practical Byzantine-resilient, yet decentralized federated learning

A thesis written by Joost Verbraeken examining the state-of-the-art and proposing Pro-Bristle, a novel technique to improve byzantine-resilience in asynchronous non-i.i.d. settings



## Abstract

We improved all key aspects of decentralized federated learning, presenting a realistic, attack-resilient solution for the first time. We envision broad use of our novel algorithms, bla bla etc. We experimentally demonstrate that Pro-Bistle is highly scalable and attack-resilient compared to state-of-the-art solutions

| Glossary | |
|---|---|
| **Machine learning** | A field that develops algorithms designed to be applied to datasets, with the main areas of focus being prediction (regression), classification, and clustering or grouping tasks[79] |
| **Distributed learning** | A type of machine learning where the workload is distributed by a master node to a cluster of dedicated and trusted slave nodes |
| **Federated learning** | A type of distributed learning where the nodes possess private data that they use to collaboratively train a machine learning without sharing their data. |
| **General Data Protection Regulation** | A legal framework that sets strong privacy-oriented guidelines for the collection and processing of personal information from individuals who live in the European Union |
| **Private Set Intersection-Cardinality** | A cryptographic technique that allows two parties to compute the cardinality of the intersection of their sets without revealing any other information[80]. |
| **Transfer learning** | A research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem[81] |
| **Exploration vs exploitation** | The problem between finding a balance between exploring new options (most of which are usually bad, but some may be very good) and exploiting the current options (which may good, but not optimal) |
| **Synchronous** | When the nodes in a network synchronize (i.e., wait for all other nodes to catch up) after each iteration |
| **Not independent and identically distributed** | When the datasets have a different probability distribution |
| **Master / slave** | Terminology used in centralized distributed learning literature to denote the node that orchestrates the learning process and aggregates intermediate results (*master*) and the nodes that train the network (*slaves*) |
| **Parameter server / client** | Terminology used in centralized federated learning literature to denote the node that orchestrates the learning process and aggregates intermediate results (*parameter server*) and the nodes that train the network (*clients*) |
| **Nodes** | A computer participating in a network |
| **Peer** | A *node* that has agreed to communicate with another *node* in a decentralized network |
| **Stragglers / laggards** | Nodes that have performed fewer iterations (lagging behind) than other nodes |
| **Convolutional Neural Network** | A type of neural networks that uses mathematical convolutions to analyze images |
| **Deep learning** | The scientific field of neural networks consisting of at least one hidden layer |
| **Neural network** | A computing system based on a collection of interconnected artificial neurons that dynamically adjust their parameters to "learn" to perform a certain task |
| **Recurrent neural network** | A type of neural networks that learns temporal sequences |
| **Stochastic gradient descent** | A variation on gradient descent based on a randomly selected subset of the data |
| **Peer-to-peer** | Direct communication between two peers in a decentralized network |
| **Data poisoning attack** | A Byzantine attack where the training data is manipulated |
| **Model poisoning attack** | A Byzantine attack where the model parameters are manipulated directly |
| **Byzantine** | Refers to the broadest type of malicious behavior, from crashes and noise in the network to carefully crafted attacks designed to subvert the system's performance |
| **Gradient Descent** | An iterative method to optimize a function by taking steps in the opposite direction of the gradient |

| | |
|---|---|
| **Gradient aggregation rule** | An algorithm that combines a set of gradients, usually in either a Byzantine-resilient, non-i.i.d.-resilient, or asynchrony resilient manner |
| **Proof of stake** | A consensus mechanism used for distributed ledgers to prove the validity of blocks based on the stake of users in the network |
| **Verifiable random function** | A function that provides a publicly verifiable proof of the correctness of its output |
| **Distributed ledger technology** | Infrastructure that enables decentralized and trusted databases by using cryptography; blockchains are the most popular type of DLT |
| **Proof of work** | A consensus mechanism used for distributed ledgers to prove the validity of blocks based on the successful calculation of a computationally expensive problem |
| **Secure multi-party computation** | Cryptography techniques used to calculate function jointly between different parties without exposing the parties' inputs |
| **Intel Software Guard Extensions** | A TEE available for Intel CPUs |
| **Trusted Execution Environment** | An isolated execution environment on processors that enables the execution of code that cannot be "spied on" by other code |
| **Continual learning** | Scientific field concerned with enabling neural networks to continuously update the model with new data distributions while retaining useful prior knowledge |
| **Lifelong learning** | See "Continual learning" |
| **Fisher information matrix** | Matrix containing the covariance of the score function in a neural network, the diagional of which is often used as a proxy for the importance of the NN's parameters |

| List of common abbreviations | |
|---|---|
| **GDPR** | General Data Protection Regulation |
| **PSI-CA** | Private Set Intersection-Cardinality |
| **Non-i.i.d.** | Not independent and identically distributed |
| **NN** | Neural Network |
| **CNN** | Convolutional Neural Network |
| **DL** | Deep Learning |
| **FL** | Federated Learning |
| **RNN** | Recurrent Beural Network |
| **SGD** | Stochastic gradient descent |
| **P2P** | Peer-to-peer |
| **GD** | Gradient Descent |
| **GAR** | Gradient Aggregation Rule |
| **PoS** | Proof of Stake |
| **VRF** | Verification Random Function |
| **DLT** | Distributed Ledger Technology |
| **SMC** | Secure Multi-party Computation |
| **Intel SGX** | Intel Software Guard Extensions |
| **TEE** | Trusted Execution Environment |
| **CL** | Continual Learning |
| **LL** | Lifelong Learning |

# Introduction

## 1.1. Motivation for federated learning

To understand the motivation for federated learning, we will first look at the purpose of traditional *machine learning* and *distributed (machine) learning*.

### Why machine learning?

A very old and well-developed branch of mathematics is statistics which is concerned with explaining and collecting insights from historical data, for example to understand consumer preferences, determine the core components of human personalities, or to illustrate how economic policies affect society. However, statistics have their limitations: it can be notoriously difficult to create a highly accurate model, the statistical techniques available to make predictions about the future are relatively limited, and the use-cases are limited to clearly specified domains (in contrast to fuzzy domains such as the generation of completely new songs from previous songs).

Therefore, machine learning gained momentum over the last few decades thanks to the discovery of several novel and powerful techniques, such as Support Vector Machines and Random Forests. These techniques are used for a wide variety of tasks such as multimedia recommendation, handwriting recognition, speech-to-text systems, digital translators, etc., but these methods rely on carefully extracted features and often yield suboptimal accuracy. Over the past decade, neural networks have gained popularity thanks to their versatility, ability to learn to extract proper features themselves, and highly effective predictions, especially when given an abundance of data [82]. Neural networks consist of a series of layers, each with a number of artificial neurons. Each neuron is connected to a number of other neurons in the previous / next layer by a connection associated with a certain weight. When a neuron is activated, it checks if the combined activation it gets from all nodes in the last layer exceeds a certain threshold (i.e., its bias) and then propagates this activation to the nodes in the next layer to which it is connected. These weights and biases are constantly being adjusted in the neural network through a process called back-propagation so that the network actually begins to "learn".

### Why distributed learning?

Training a neural network on a single machine is possible when the amount of data is relatively limited. However, for more complex applications (such as self-driving cars, image recognition, or music generation) the amount of training data required can easily exceed the maximum capacity of a single machine. Verbraeken et al. [83] describe in detail how a new scientific field called *Distributed Learning* aims to distribute the training data and/or the neural network across many computers often implemented by combining an large number of *slave* nodes that perform the calculations given by a *master* node (often called the *parameter server*). The master node communicates with the slave nodes, iteratively combines their results, and updates the individual slave nodes with the result. This technique gained rapid popularity and is nowadays the backbone of most industrial-grade machine-learning implementations [84] (although there is also a multitude of other distributed learning architectures, each with their own advantages and disadvantages [83]).

### Why federated learning?

Although distributed learning is highly effective in teaching neural networks to accomplish complex tasks, it still depends on high quantities of data to get the most accurate results. The data to train popular neural networks often comes from smartphones, which on one hand produce massive quantities of data that allow for improved representation and generalization of machine-learning models, but on the other hand pose a significant problem for three main reasons: (a) transmitting all kinds of data generated by smartphones over the Internet consumes a lot of bandwidth, (b) training a neural network on data generated by billions of smartphones is computationally extremely intensive for a single *master* node, and (c) transmitting potentially sensitive information over the internet to the cloud raises privacy concerns, and is in certain cases not even allowed by several regulations such as the US HIPAA laws [85] and Europe's *GDPR* law [86]."

These challenges prompted the development of a new type of distributed learning called *federated learning*, where smartphones update the neural network with their private data on-device and send the updated model back to the server instead of their data [51]. In the words of McMahan et al, "Federated Learning brings the code to the data, instead of the data to the code" [87]. From this perspective, federated learning is closely related to Mobile Edge Computing (MEC)

in the sense that computations are pushed to the edge of the network to reduce bandwidth consumption and improve privacy.

Thanks to these advantages, federated learning is now used for a wide variety of applications including next-word-prediction on keyboards such as Gboard [88-94], "wake word detection which enables voice assisting apps to detect wake word without risking exposure of sensitive user data" [95], speech recognition [96], wireless communications [97, 98], security applications (such as malware classification [99], human activity recognition [100], anomaly detection [101], and intrusion detection [102]), transport applications (such as data sharing between self-driving cars [103-105], preventing data leakage [106], traffic flow prediction [107], and the detection of attacks in aerial vehicles [108]), object detection [109], and health applications [110-114].

## 1.2. Challenges and design principles

We will first limit the scope of the thesis to keep the project manageable: we will focus on arguably the most popular type of machine learning, namely supervised learning. This is a type of machine learning where the model must learn the correlation between the input data and the corresponding labels based on a set of training samples. From the two main types of supervised learning (namely classification and regression) we will look at classification problems.

Given that we focus on classification problems, federated learning environments have a number of notable characteristics [87, 115]:

- **Massively distributed**: the total number of nodes can easily be in the order of millions [116].
- **Unbalanced / non-i.i.d. data**: non-i.i.d. or not "independent and identically distributed" means that different peers may possess different classes distributed in different ratios.
- **Unreliable**: since federated learning is often applied to smartphones, the nodes may go offline/online at any moment and the network connection may be slow.

However, we want to build a system that not only seeks to properly address the challenges imposed by the characteristics mentioned above but is also practical to be used in a realistic environment. Therefore, we want to design the system around four additional principles:

- **Decentralized**: eliminate a single point of failure prone to crashes and hacks.
- **Byzantine-resilient**: Byzantine attackers must not be able to subvert the model's performance or inject backdoors into the model (see Section 2.1).
- **Asynchronous**: synchronous systems are dependent on the slowest node in the system (*straggler*); asynchronous systems may help to mitigate stragglers.
- **Communication-efficient**: the bandwidth consumed by the system should be as low as possible to reduce costs and increase performance.

In the literature, addressing privacy challenges is also often mentioned as an important element of federated systems. We explicitly leave privacy concerns out of the scope of this thesis, as the solutions for improving user privacy do not appear to be correlated with the solutions to the aforementioned problems.

Whereas the first three characteristics of federated learning environments have already elaborated upon in detail in [87, 115], we will now discuss the latter four additional principles in more detail:

### Decentralized

Whereas in distributed learning the developer owns all computers and thus has a system where the slave nodes do exactly what the master node tells them to do, the devices in federated systems are owned by their users rather than the developer and may or may not do what they are asked to do. Practically all federated learning systems employ a centralized architecture characterized by a central trusted authority [87], often called a *parameter server*, that communicates with all *clients* (as they are called in a centralized federated setting; these devices are often smartphones). The machine learning process is as follows: (1) definition of the ML model (e.g., a CNN or RNN) by the developer in terms of hyperparameters, (2) distribution of the model by the server to the clients, (3) local training of the model by each client, (4) aggregation of all models by the server, (5) iteratively repeating steps 2, 3, and 4 [116]. The training process may stop when a sufficiently high accuracy is obtained or may be trained continuously as more data becomes available to the clients.

A parameter server offers several significant advantages: it can reduce the total bandwidth used by the system (since clients typically only communicate with a single server), it can employ more sophisticated Byzantine-resilience measures (since a single server node that receives all models can compare all models received with each other), and gradient aggregation updates can be propagated immediately (when a new type of attack subverted the model's performance, the developer can just load a backup and add a proper defense mechanism against that attack).

Unfortunately, such a centralized architecture also has several significant drawbacks [116-118]. The parameter server is not only a single point of failure susceptible to crashes or hacks, but it may also become a performance bottleneck when there are too many devices participating in the network[117]. A parameter server typically communicates with all clients in the network, which contrasts sharply with decentralized networks where each node typically communicates (called *gossiping* in decentralized networks) with only a small number of other nodes (called *peers* in decentralized networks), thus incurring no additional communication costs when an additional node joins the system. This problem accelerated research that aims at completely removing the parameter server and training the network in a decentralized manner [117, 119-121], with each node both training and aggregating incoming parameters to learn the model [122]. This led to a new generation of decentralized learning methods that achieve similar accuracy as state-of-the-art centralized methods while being significantly more scalable and robust[123].

### Byzantine-resilient

Even the most efficient and stable decentralized federated learning system is worthless for practical applications when the model can be subverted by Byzantine nodes, with *Byzantine* referring to the broadest class of system component errors. A Byzantine model may be created accidentally (e.g., due to a crash, faulty sensor, computation error, noisy transmission, node that is lagging behind, *non-i.i.d.* data, etc.; all of which the probability to occur increases with the number of nodes [124]) or created on purpose (e.g., data poisoning or model poisoning attacks; more on this in Section 2.1). This scenario, where the nodes do not know which of the other nodes are benign or corrupt, is the infamous "Byzantine Generals Problem" [125]. Without a Byzantine Fault Tolerance (BFT) mechanism, even a single malicious node that uses only moderate values to make its actions hard to detect can significantly degrade the performance of the federated model [126, 127].

Unfortunately, it is relatively easy to initiate a simple poisoning attack where a node intentionally sends incorrectly updated parameters for three main reasons [128]: (1) authentication mechanisms are often not feasible because federated systems often span across countries, (2) because the whole goal of federated learning is to keep the training data private, it is impossible to verify the reliability of the training data, and (3) in real-world situations that dataset is often (very) *non-i.i.d.* making it challenging to distinguish between an attack and an unusual data class.

### Asynchronous

Distributed learning can happen either in *synchronous* rounds or in an *asynchronous* manner. When the system is synchronous, all nodes wait on each other until all nodes have received each other's update and caught up with training. In asynchronous systems, the nodes do not wait for each other, enabling faster nodes to perform more iterations than slower nodes. Although synchronous algorithms may seem to be the logical choice for federated learning (illustrated by the fact that the first federated learning protocol FedAvg [87] and influential subsequent research such as [129] were synchronous), there are a number of limitations as summed up by [130]:

- **Unreliable**: some devices assumed to participate in a synchronous round may fail randomly due to the volatile nature of the end-devices (similar to the *Unreliable* characteristic of FL environments mentioned above).
- **Waiting for next synchronization**: devices that just joined the network and are ready / willing to participate will have to wait until the start of the next synchronization and are thus under-utilized. Additionally, faster devices have to wait for slower devices called *stragglers* or *laggards* which are common given the very heterogenous nature of federated systems in terms of node hardware capabilities.
- **Too slow**: when a device is too slow to finish its iteration before the next synchronization, it has to overwrite its local model with the new global model and all of its progress is lost.

For this reason, numerous asynchronous approaches have been proposed [24-28, 115, 116, 130, 131]. These methods either overprovision clients and then accept the first update(s), dynamically update the synchronization time or amount of work per node, or use weighted averaging based on the iteration number of incoming model updates.

### Communication efficient

Since the total number of nodes can easily be on the order of millions [116] and a typical neural network can easily exceed a few million nodes, the total bandwidth used by the system can be massive. This is all the more true for decentralized systems, where nodes often communicate not only with a single server node, but with several other nodes simultaneously. It is important to optimize the bandwidth used to transmit the models to improve the time it takes to train the network and to minimize the network communication costs.

## *1.3. Research questions and key contributions*

The main research question we want to answer is as follows:

- How can we create a decentralized federated learning system suitable for real-world scenarios?

Based on the challenges and design principles stipulated above, we formulated the following sub-research questions:

1. What kind of Byzantine-attacks exist and how do they work?
2. How can we achieve a decentralized, reliable, massively distributed federated learning system?
3. How can we make a federated learning system Byzantine-resilient?
4. How can we make a federated learning system asynchrony-resilient?
5. How can we make a federated learning system non-i.i.d.-resilient?
6. How can we make a federated learning system communication-efficient?

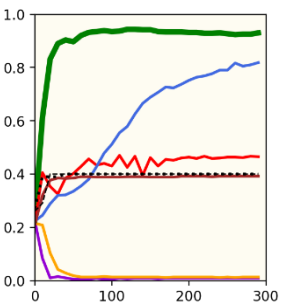
We grouped the decentralization, reliability, and massively distributed aspects together since these problems are strongly interconnected: a decentralized system is typically also highly scalable because there is no single bottleneck, and situations where a node suddenly stops communicating are also very common in decentralized networks.

Answering these research questions is highly non-trivial, as distributing a computation over many peers induces a substantial risk of local crashes, computation errors, stale processes, and biased local datasets. A recent survey paper called dealing with non-i.i.d. data a key challenge [132], not to mention dealing with non-i.i.d. data in a highly Byzantine environment (>50% of the nodes is malicious). As a consequence, literature on this topic is sparse. All of the challenges and design principles outlined above are addressed by the main contribution of this work: a new and powerful GAR (Gradient Aggregation Rule) named Pro-Bristle (Practical yet RObust Byzantine-Resilient decentralIzed StochasTic federated LEarning). Pro-Bristle is (a) highly scalable, (b) able to handle non-i.i.d. data, (c) reliable, (d) decentralized, (e) Byzantine-resilient, (f) asynchronous, and (g) communication-efficient. Figure 1 visualizes how the challenges and solutions that together constitute Pro-Bristle relate to each other.

The key contributions of this thesis can be summarized as follows:

- We propose to use a combination between per class performance-based filtering, SRA private-set intersection cardinality, CWR*, deep transfer learning, and a carefully crafted weighted averaging function to achieve a high degree of Byzantine-resilience in non-i.i.d. environments.
- We use per class performance-based filtering, a buffer of recent models, and an exploration vs exploitation strategy to account for highly asynchronous situations.
- We evaluate Pro-Bristle and five other GARs in a large variety of environments, with varying datasets, degrees of asynchrony, attacks, number of attackers, degrees of non-i.i.d.-ness, usage of transfer learning, and communication protocols.
- We call for a clear distinction between the integration of familiar classes (classes where the peer has sufficient samples) and the integration of foreign classes (classes where the peer has insufficient samples).

To illustrate the key contribution, we highlight a figure shown in Section 10. In this figure, the performance of different GARs in a mildly Byzantine (label-flip attack) and somewhat non-i.i.d. environment is illustrated. The thick green line is the performance of Pro-Bristle, which clearly outperforms all other GARs, despite consuming only a fraction of the bandwidth (details in are given in Section 10).



## 1.4. Thesis overview

First, in Section 0 we aim to give the reader an idea of what types of Byzantine attacks exist and how they work. After we better understand the various types of Byzantine attacks, Sections 3, 4, 5, 8, and 7 summarize for each research question stated in Section 1.3 (visualized in Figure 4) the literature written about the topic and the approach that we take with Pro-Bristle. Most attention is devoted to Section 4 which explicates all kinds of Byzantine-resilient defense mechanisms, since properly defending against Byzantine models is the main contribution of this thesis. In Section 8, we bring all the components that we described in the other sections together and explain how they work together to achieve superior performance in highly Byzantine non-i.i.d. environments. Section 9 describes in detail how we conducted the experiments and how the software was implemented including all Byzantine attacks. Section 10 contains and discusses all results. We conclude by discussing the most interesting directions for future research and the biggest issues that we encountered during the development of Pro-Bristle in Section 11, and the main insights that we gained in Section 12.

| Section | Challenges | Solutions |
|---|---|---|
| 3 | Decentralization | |
| 3 | Reliability | Decentralized P2P gossip learning |
| 3 | Massively distributed | Distance-based filter |
| 4 | Byzantine-resilience | Sigmoid weighted averaging |
| 5 | Async: received a stale model | Per class performance-based filter |
| 6 | Non-i.i.d. data | CWR* |
| 7 | Communication-efficient | PSI-CA (Private-Set Intersection Cardinality)<br>Deep transfer learning<br>Model compression |
| 5 | Async: received too few models | Model buffer |
| 5 | Async: received a too good model | Exploration vs exploitation strategy |

**FIGURE 1.** MAPPING OF CHALLENGES ADDRESSED BY EACH SOLUTION

## 2. Introducing the Byzantine attacks

Before we take a look at the wide variety of Byzantine-resilient defenses, we will first consider what kind of Byzantine attacks exist and how they work. We will segment these attacks along two dimensions, namely whether they are untargeted or targeted, and whether they are model poisoning or data poisoning attacks. At the end of the section, we present a table comparing the GARs.

### 2.1. Related research

A very popular (and simple) federated learning technique is to combine the model vectors by simply taking the 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 [55].

Byzantine attacks can be classified as a data poisoning or a model poisoning attack, and as an untargeted or a targeted attack based on certain characteristics shown in Figure 2.

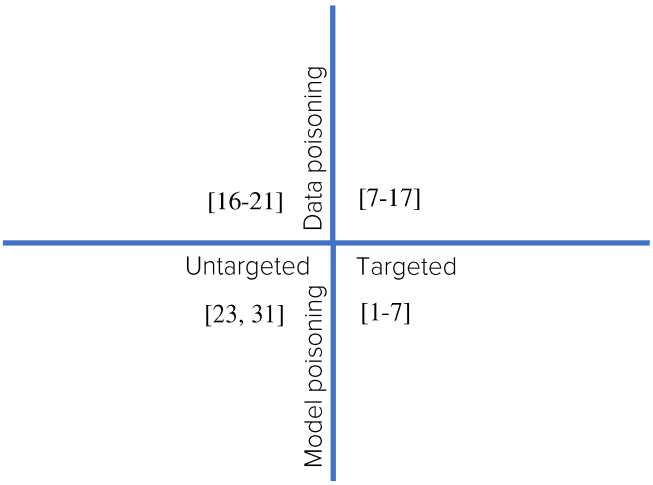



**FIGURE 2. SEGMENTATION OF BYZANTINE ATTACKS ALONG TWO DIMENSIONS**

#### Data poisoning vs model poisoning attacks

Data poisoning attacks such as [7-21] attempt to subvert the performance of the learned model to such an extent that the model becomes worthless by training the network with dirty samples. They were introduced by Gu et al. [12] to destroy support vector machines and later extended to many other ML algorithms including neural networks. Without proper Byzantine-resilient defense mechanisms, a malicious agent can relatively easily manipulate the global model. The best-researched type of attack is a *convergence-prevention attack* [4] where the attacker wants to prevent the network from converging and reduce its accuracy to such an extent that the model becomes utterly ineffective indiscriminately for testing examples. One might think that sending completely random numbers is 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 [133] (in fact, adding noise to the parameters is a popular method called *differential privacy* that is used to improve the user's privacy[134-136]). A scenario where a malicious agent injects malicious data into a benign client's dataset (better known as a *data injection attack*) is also considered data poisoning. Another notable example of a data poisoning attack is a *label-flip attack*, where the labels of two or more classes are changed [1, 16, 17].

While data poisoning attacks are based on the manipulation of training data, model poisoning attacks (introduced by [1]) such as [1-7, 23, 31] manipulate the model's parameters (usually the weights and biases) before sending it to other nodes. Consequently, every data poisoning attack can be imitated with a model poisoning attack [132], but model poisoning attacks give the attacker full control over every single parameter and can thus be much more effective, as recent research has shown [2, 55, 56]. They 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 [1]. However, there are also simple model poisoning attack, such as the *Gaussian attack*, where some of the gradient vectors are replaced by random vectors sampled from a Gaussian distribution with large variances.

#### Untargeted vs targeted attacks

Another way to classify Byzantine attacks, as done by [132], is to group them into untargeted and targeted attacks (also known as poisoning availability attacks and poisoning integrity attacks respectively [137]). Whereas untargeted attacks such as [1, 3, 15-21, 23, 31, 56, 137-139] aim to prevent convergence and reduce the global model's accuracy [31, 56], targeted attacks such as [1-17] aim to alter the model's behavior in specific situations while keeping the total

accuracy as high as possible to mislead Byzantine-defense mechanisms [1, 2]. Without proper defense mechanisms, federated learning is susceptible to both untargeted and targeted attacks [16].

Targeted attacks are also sometimes referred to as (semantic) backdoors, Trojan threats, or stealth attacks, where the attacker can target either a single class (a *label-flip attack*) or a class of samples (e.g., an almost invisible attacker-chosen pattern of pixels, i.e., a *trigger attack*) causing an image to be classified incorrectly. A particularly effective attack is described by Bhagoji et al. [5] 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 [7] who note that all backdoor attacks until then used embeddings of the same global trigger pattern for all Byzantine parties, called *centralized backdoor attacks* by the authors. They then propose *distributed backdoor attacks (DBA)* where the global trigger pattern is decomposed into local patterns and 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, [4] shows that targeted model poisoning attacks can become both significantly more effective and harder to detect when adversaries are able to collude.

As mentioned in Section Byzantine-resilient, a little random noise can actually improve the convergence of stochastic gradient descent. That could lead one to think that simply eliminating models with large deviations might be an effective defense mechanism. However, Baruch et al. [4] show that this assumption is incorrect and propose another powerful attack "capable of defeating all state-of-the-art defenses" based on injecting values that are just within the perturbation range (the range of values that the Byzantine-defense mechanism permits).

Targeted attacks are hard to detect, because the accuracy of the model does not necessarily have to be impacted for any of the samples that any peer has available, but only for samples with, for example, a specific pattern that no-one but the attacker knows about. More specifically, detecting backdoors in a model is an NP-hard problem, by a reduction from 3-SAT [32], and unlikely to be detected using gradient based techniques. To illustrate this, Wang et al. [32] 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 targeted model poisoning attacks only need to modify a small part of the model [132], they look quite similar to benign updates and require fewer adversaries than untargeted attacks, as they are already effective under certain conditions even on a single-shot attack[1].

## 2.2. Attacks used in this research

To evaluate the Byzantine-resilience of the GARs discussed in this thesis, we setup several Byzantine agents that aim to subvert the model. We selected the following attacks to be used in the experiments:

### Untargeted data poisoning: All-label-flip attack [1]

As described in Section 2.1, a popular data poisoning attack is the label-flip attack where the labels of two or more classes are changed [1, 16, 17]. For this thesis, we evaluate both a label-flip attack where 2 labels are flipped, and a label-flip attack where all labels are flipped. The all-label-flip attack is an untargeted data poisoning attack since it aims to completely destroy the model's performance by manipulating the labels of the training data.

### Targeted data poisoning: 2-label-flip attack [1]

Similar to the all-label-flip attack, the 2-label-flip attack is a data poisoning attack, but the 2-label-flip is targeted: performance on all non-flipped classes is unaffected, making the attacks become weaker, but also harder to detect.

### Untargeted model poisoning: Additive noise attack [23]

As explained in Section 2.1, simply sending random noise with a small variance is ineffective to prevent convergence, because the mean of the noise equals 0. When the noise has a larger variance, it can indeed prevent convergence, but also that makes the noise attack easier to detect [23]. Centering the noise around a value slightly different from 0 allows the attack to prevent convergence despite low variance, but since benign updates are always centered around 0, this attack can be easily detected. We opt for a variant where half of the parameters are set to noise centered around a value just below 0, and the other half of the parameters is set to noise centered around a value just above 0.

### Targeted model poisoning: Krum attack [56]

Whereas data poisoning attacks generally do not make assumptions about the GARs used, model poisoning attacks often target a specific GAR. Fang et al. presents in [56] an effective attack against Krum by iteratively sending an attack vector that will be *just* accepted by Krum whilst inflicting maximum damage to the peer's model.

### Targeted model poisoning: Trimmed Mean attack [56]

In the same paper as the aforementioned Krum attack, Fang et al. also describe a model poisoning attack against the Trimmed Mean GAR. The attack determines the gradient direction for each parameter of the model and then creates an attack vector that is exactly the opposite direction, scaled per parameter depending on the values of the other benign peers.

# 3. Achieving decentralized, reliable, massively distributed federated learning

## 3.1. Related research

In this thesis we will group the decentralization, reliability, and massively distributed aspects together, as these problems are interconnected. A decentralized system is typically also very scalable because there is no single bottleneck, and situations where a node suddenly stops communicating are also very common in decentralized networks. Since we are interested in decentralized *federated learning*, we will first give the reader a primer into federated learning, and then discuss the literature written on decentralized federated learning systems, how these systems deal with unreliable nodes and massively distributed settings, and finally present our own solution to achieve this goal.

### 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 $\theta^*$:

$$\theta^* = \underset{\theta}{arg\,min} \underset{(x,y) \in D}{\mathbb{E}} \ell(f_\theta(x), y) \tag{1}$$

Where $\theta$ is the model, $D$ is a distribution on $\mathrm{X} \times \mathrm{Y}$ (with X denoting the data and Y denoting the corresponding labels), and $\ell(\theta; i)$ is the loss of model $\theta$ on dataset instance $i$. This loss function is a proxy for the actual error to be minimized, generally the negative log likelihood of the ground truth class in the case of a classification problem.

This optimization problem is known as *risk minimization*, but unfortunately 0solving this problem is intractable for more complex models. Therefore, a technique called Empirical Risk Minimization (ERM) is commonly used where we take an empirically obtained dataset $M$, i.i.d. sampled from $D$. Then we can obtain an estimate of the optimal model by calculating:

$$\theta_n = \underset{\theta}{arg\,min} \frac{1}{|M|} \sum_{(x,y) \in M} \ell(f_\theta(x), y) \tag{2}$$

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:

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

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) [140] where a subset (a *minibatch*) of the dataset is selected to update the parameters in a particular iteration [141, 142]. 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 to escape local minima. An important requirement for SGD to converge is that each minibatch is an unbiased sample of the true distribution, which is usually achieved through uniform random sampling [143].

The most straight-forward way to apply stochastic gradient descent in a distributed or federated setting is to use a single master node (*parameter server*) that distributes and aggregates the global model $w = \bigcup_{i \in \mathcal{N}} w_i$, and a number of slave nodes that train the model that they obtained from the master node and send the result back to the master node [144, 145]. A common assumption is that all slave nodes are benign, which cannot always be guaranteed and is therefore addressed in the next section in this thesis (Section 4).

In distributed machine learning, the most trivial implementation is called Bulk Synchronous Parallel [146] and in federated learning FedAvg [87]. FedAvg simply aggregates the models owned by the peers by coordinate-wise weighted averaging. It was introduced by Google [43] and is still extensively researched from both an applied and theoretical perspective [116]. For non-convex parameter spaces, averaging models can yield poor results as good models can converge in different directions, but fortunately it seems that sufficiently over-parameterized neural networks are mostly convex and not very prone to bad local minima [147-149]. The pseudocode for FedAvg is given in Algorithm 1.

**Input:** Local minibatch size $B$, set of slaves $K$, number of slaves per iteration $k$, #local epochs $E$, learning rate $\eta$, set of local datasets $D$

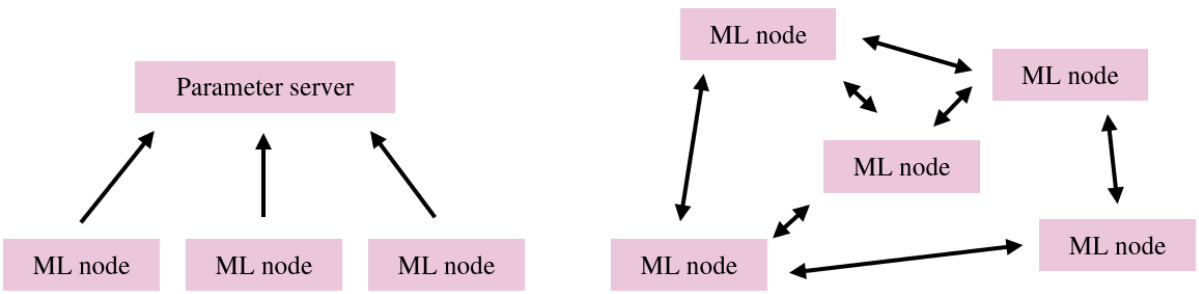
1. **Server:**
2. Initialize $w_0$
3. **For** each round $t = 0, 1, 2, \dots$ **do**
4. $S(t) \leftarrow$ (random set of $m$ clients from $K$)
5. **For** each client $k \in S_i$ **in parallel do**
6. $w_k(t+1) \leftarrow ClientUpdate\big(k, w(t)\big)$
7. **End for**
8. $w(t+1) \leftarrow \frac{1}{\sum_{k \in S(t)} D_k} \sum_{k \in S(t)} |D_k| w_k(t+1)$
9. **End for**
10.
11. **Client** $k$:
12. $\mathcal{B}_k \leftarrow$ (split $D_k$ into batches of size $B$)
13. **For** each local epoch $e = 0, 1, \dots, E$ **do**
14. **For** batch $b \in \mathcal{B}_k$ **do**
15. $w \leftarrow w - \eta \nabla l(w; b)$
16. **End for**
17. **End for**
18. Return $w$ to server

**ALGORITHM 1** BASIC CENTRALIZED FEDERATED LEARNING PSEUDOCODE EXECUTED ON THE SERVER AND EACH CLIENT

## Decentralized federated learning

As mentioned in Section 1.2, centralized federated learning systems where all clients communicate with a single parameter server have several drawbacks, most notably the fact that a single parameter server is a single point of failure, susceptible to hacks and crashes, and can be a serious bottleneck for the entire system. By removing this parameter server and thus completely decentralizing the entire system, these problems can be alleviated. Such a decentralized system is also called a *peer-to-peer* (P2P) system, and the communication between the nodes is called *gossiping* [150]. Another commonly used term for decentralized federated learning is gossip learning[123].

Decentralized learning works in much the same way as distributed learning but differs in a number of aspects. Algorithm 2 shows the code that each node executes. At each iteration, the node trains its local model (typically implemented as a stochastic gradient descent step), integrates it with all models received so far from peers during that iteration (typically implemented by unweighted averaging), and then sends it to a subset of its peers [151]. Note that these iterations are not synchronized. In practice, there are many variations where the models that are received or the transmission to its peers does not happen every iteration, where the transmission to other peers is spread out over multiple iterations, or where compression and / or Byzantine-resiliency measures are applied during the iteration [132]. These peers are selected by a so-called *peer-sampling service* that chooses a number of random peers, uses round-robin, a torus, a ring, or some other method to select a set of peers. This contrasts with centralized architectures where communication only happens with a central server. The total set of peers is maintained by a background service that typically performs random walks and sends introduction requests to known peers to keep track of which peers are still alive, and to discover




(a) Single, centralized parameter server (b) Decentralized peer-to-peer

**FIGURE 3.** THE DIFFERENCE IN THE ARCHITECTURE BETWEEN CENTRALIZED FEDERATED LEARNING AND GOSSIP LEARNING

new peers. When the total set of peers is kept relatively small, it does not necessarily have a significant adverse effect on the convergence rate in practice, as denser networks also incur extra communication delays [152].

**Input:** Minibatch size $B$, set of peers $P$, #local epochs $E$, learning rate $\eta$, local dataset $D$

1. **Node:**
2. Initialize $w_0$
3. $D' \leftarrow$ (split $D$ into batches of size $B$)
4. **For** each epoch $e = 0, 1, \ldots, E$ **do**
5. **For** batch $b \in D'$ **do**
6. $w(t+1) \leftarrow w(t) - \eta \nabla l(w; b)$
7. $w(t+1) \leftarrow$ (merge $w(t+1)$ with received models from peers)
8. **For** peer $p \in P$ **do**
9. Send $w$ to $p$
10. **End for**
11. **End for**
12. **End for**

**ALGORITHM 2** BASIC DECENTRALIZED FEDERATED LEARNING PSEUDOCODE EXECUTED ON EACH NODE

## *Our solution*

Motivated by the very recent results of Hegedűs et al. who showed the gossip learning can perform on par or even outperform centralized federated learning [153], we will also use *gossiping* to enable a massively distributed number of nodes learn together in a decentralized, robust, and scalable way. With gossiping we mean that every node sends at every iteration to a few other nodes an update [29, 153]. Whenever possible, the nodes send their updated model to nodes that are different from the nodes from which the node received an update to propagate the updates faster through the system. Gossiping also makes the fact that nodes are unreliable less relevant, since gossiping happens with "a random node"; if some node happens to be offline, the nodes will just choose other nodes to gossip with. Note that due to the nature of gossiping, every node has a (slightly) different model.

Since the Byzantine-resilience component of Pro-Bristle depends on overlap between the datasets of peers, we use PSI-CA to restrict the sets of peers to peers whose dataset overlaps with the dataset of the given node (see Section 4.2). Since a peer may, despite this restriction, receive a number of models in a massively distributed gossiping system that is too large for the peer to handle, we also use a computationally cheap distance-based filter to filter out models that are likely to be Byzantine. The distance-based filter calculates the Euclidean distance between the peer's own model and all models that it received, which is not very expensive because we only share the weights of the output layer, see CWR* in Section 6.2). Only the most promising models are evaluated more extensively with more a computationally expensive algorithm (see *per-class performance-based filtering* in Section 4.2 and *exploration vs exploitation* in Section 5.2).

Because all nodes "randomly" gossip with each other, there is no longer a single global state of the model, but when we assume that a communication path exists between each pair of nodes, the local models will gradually reach consensus and converge (see Section 4.2).

# 4. Achieving Byzantine-resilience

## 4.1. Related research

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), categorize these methods into 5 distinct categories, and subsequently how we mitigate Byzantine attacks with Pro-Bristle.

### Byzantine-resilient defenses

Byzantine resilience can be divided into *weak* and *strong Byzantine resilience* [154]. 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 [155] 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 and 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 [156-158]. 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 [159] 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 ruin 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 (based on the calculation of some kind of distance between potential malicious attack vectors and some other vector(s), usually efficient but also vulnerable to elaborately designed Byzantine attacks [4, 56]) and performance-based GARs (based on testing the accuracy of a potentially malicious model on a small representative dataset, which is usually computationally quite intensive and clearly dependent on the availability of a test dataset but also usually quite effective) [33]. Another way of segmenting the these algorithms is based on whether or not they are centralized (dependent on a central parameter server) or decentralized, by their degree of dimensional Byzantine resilience [33] (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 1.

#### *Distance-based screening*

Screening potentially malicious incoming model updates against their distance to the peer's own trusted model is by far the most popular method to evade Byzantine attacks, which should come as no surprise. They are often quite efficient, do not depend on an additional dataset, special hardware features, or an additional server, and they provide 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, they perform poorly compared to more advanced attacks [160]. 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, gradient updates can differ significantly in a non-i.i.d. environment between nodes which results in large distances, causing distance-based GARs to reject these updates as outliers. Moreover, when the peer's own model is extremely stale, all incoming models are considered outliers, making it hard for the stale model to catch up. Therefore, in the other sections other methods that are computationally much more expensive are evaluated that can be more effective than distance-based GARs, especially in non-i.i.d. and asynchronous settings.

Outlier detection in non-distributed settings has been studied extensively for a long time [161], generally in order to purify the data from poisoned or otherwise aberrant data [162]. Much progress has been made in recent years in terms of improved accuracy in high-dimensional settings [163-165]. For example, [14] uses a clustering technique to measure the difference between benign and malicious updates. However, these techniques are not suited for the distributed setting on which we are focusing.

**Krum** [126] 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, in theory the

performance should not degrade too much as it is close to all other models. Despite theoretical guarantees for the convergence for certain objective functions, Krum appears to perform poorly compared to other algorithms [57] and often converges to an ineffective model [31]. 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 [31]. Baruch et al. [4] elaborate upon this insight and argues that since only a single model is selected and even the best benign worker will have a few parameters far from the mean, the GAR 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 an average of $m$ local gradients obtained by Krum.

**CTM / CM** [55, 166, 167] 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. **Coordinate-wise Median (CM)** or Marginal Median takes the median in each dimension. CM does not need at least *2b + 1* values like CTM, but it does incur a performance hit because each dimension has to be sorted to get the median.

**GeoMed / MeaMed** [33] were compared by Xie et al. [33] under non-convex settings with Krum. The **Geometric Median(GeoMed)** is defined as [33, 126, 127]:

$$\underset{v \in \mathbb{R}^d}{argmin} \sum_{i=1}^{n} \|v - \tilde{v}_i\| \tag{4}$$

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 [34].

**(Geometric) Median of Means** [127, 168-172] is a variant of 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. [127] extend the techniques described in [172] with arbitrary/adversarial outliers, but they only consider strongly convex losses which they then try to remedy by using mini batches. However, their algorithm fails even when there is only a single Byzantine node in each mini-batch and is thus not very reliable.

**Bulyan** [31] 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 [154].

**SignSGD** [35] was developed to reduce the communication necessary for the aforementioned GARs by transmitting only the sign of every dimension of the gradient at every 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 [36]. However, one of these conditions is that the data is distributed non-i.i.d., which is typically not the case in federated learning environments[58]. Sohn et al. make SignSGD more robust against MITM-attacks, but do not address the case where nodes themselves are malicious[37].

**RSA** [173], confusingly having the same name as the cryptosystem, is in contrast to the methods mentioned until now able to handle heterogenous datasets in a Byzantine distributed setting. It aims to prevent incorrect gradient aggregation by letting every node store and update a local version of the global model which are then aggregated at the server by means of an $\ell_p$-norm regularization term which regularizes 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 [60], BRIDGE [61], and some extension of BRIDGE [62].

**ByRDiE** [60] recognizes that the CM algorithms described before 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 and the update step depends on the updates of other coordinates [57]. Therefore, Yang et al. [61] present BRIDGE which combines CTM with SGD to achieve decentralized Byzantine-resilience with significantly less network communication for high dimensional problems. The same authors later showed better performance for BRIDGE than for CTM [57], 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 $4\ x\ \#\max\ \_byzantine\_nodes + 1 = 4\ x\ 2 + 1 = 9;\quad 9\ /\ 20\ nodes = 0.45$ connection ratio between the nodes to evaluate CTM. [62] shows that BRIDGE's can be improved by adding a total variation (TV) norm penalty to allow some outliers to be able to handle non-i.i.d. data. This likely reduces the ability of the algorithm to defend against noise attacks, but unfortunately the authors have conveniently omitted these results from the paper. [63] also builds upon BRIDGE and extends the solution to non-i.i.d. settings, but does this by re-introducing the central server that we wanted to omit in a decentralized setting in the first place (more information about non-i.i.d. approaches will be discussed in Section 3).

One of the most recent articles about this topic is [38] 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 [59] 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.

### *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., in a non-i.i.d. or highly asynchronous environment) and also allow an attacker to let the model drift towards a bad solution. Performance-based solution such as [10, 39, 40, 64, 162, 174] detect malicious models based on a negative 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 in the presence of a large number of adversaries [40, 42, 47, 64, 69, 175]. We want to seriously criticize the paper written by Zhao et al. [40] 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 2.1). 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** [174] **/ TRIM** [137] 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 maximizes the accuracy and is, 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. [56] converted and applied RONI and TRIM to a federated setting and found that in a federated setting RONI gives slightly better performance.

**Zeno** [41] **/ Zeno++** [42] were introduced by Xie et al. for synchronous environment and asynchronous environments respectively. Both use a centralized oracle that estimates based on a validation dataset the true gradient and only keeps the $k$ gradients most similar to this estimation. The performance of both GARs is quite good according to [57], but they depend on a centralized parameter server and need access to a sufficiently large unbiased validation dataset.

**PDGAN** [65] 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 [5, 66], and are, in that regard, also a "highly impactful and prioritized" [176] attack in their own right.

**Mozi** [160] was an important inspiration for this 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).

### *Pruning*

Since backdoor attacks (see Section 2.1) are extremely challenging to detect, an entirely different class of GARs called "pruning" defenses has been proposed, specifically aimed at preventing these backdoor attacks [44, 67, 177]. Pruning defenses use a representative subset of the global dataset (partially violating the FL assumption [132]) 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 [12].

Unfortunately, even when these inactive neurons are removed from the model, more adaptive poisoning attacks are still possible [68]. After all, the boundary between a neuron being unused or being actively used is vague.

There are several other methods aimed at detecting backdoor [39, 48, 67, 76, 77, 177-180], 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 setting) or access to a holdout set of similarly distributed data (which cannot help defend against more sophisticated model poisoning attacks as discussed in Section 2.1).

### *Behavioral-based*

**FoolsGold** [16] is an algorithm to detect and reject 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. [40] 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 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 per node (such as [69]) does not work well, because a Byzantine attacker can first build up an excellent reputation, and then suddenly completely ruin the model, 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 [45, 70-73, 103, 181-190], usually a blockchain. This ledger can also be used to save global model parameters to enhance the system security [183, 186].

Kang et al. introduced in [191] reputation as a means to determine the reliability of every node and subsequently proposed a GAR based on these reliability scores[192], using RONI to calculate reputations in i.i.d. environments and FoolsGold to calculate reputations in non-i.i.d. environments. For this to work, the authors (implicitly) assume an environment where nodes have a strong identity and where there are many different parameter servers learning different tasks that share reputation opinions of nodes on a public blockchain.

Zhao et al. [74] 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 [57]) 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 very limited.

Whereas all former approaches assume that the individual workers might be Byzantine, [193] 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 [45] first splits the nodes into a set of workers that use their data to train the model for a single iteration and into a voting committee that votes for the best proposals and stores this information on a blockchain (similar to [70, 194]. The voting committee is usually selected on the basis of *Proof of Stake* (Pos) and *Verifiable Random Functions* (VRFs)). The method is fully decentralized, but only defends against a factor of $1/3^{rd}$ Byzantine workers. The technique is hardly scalable to a large number of nodes, because every node in the voting committee has to evaluate every single update, and because either all voting committee nodes are waiting for the workers to be finished or vice versa, a significant amount of time is spent idling for each node.

Although the blockchain papers mentioned above are of good 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, in [103] the authors 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 make take advantage of DLT's strengths is [181] where 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 GAR and includes 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 [73, 185, 188-192, 195-207] and Stackelberg game methods [195, 196, 208, 209], but since these methods are not intended as defense 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 applications, such as Gboard, Captcha, or Google Fit.

### *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 [46, 73, 75, 210, 211]. 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 [210]. Sabt et al. also discuss how TEEs can be used as a defense technique [211], whose insights are later used to create a generic framework that can be used to integrate TEEs in a federated learning environment [46, 75]. Weng et al. [73] 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 environment. 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 [47]. 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 [23]. 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** [34] is a well-cited example of a final type of GARs we would like to highlight: GARs based on replicating the same training over multiple nodes [34, 37, 43, 212]. When nodes are benign, they will (under several assumptions) report the same results. 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, which makes the algorithm inappropriate for a decentralized federated environment. For example, 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 $5\ x\ 2 + 1 = 11$ times.

Overview of the most notable GARs

| GAR | Condition on [M, b] | No prior information about #attackers | Non-i.i.d. resilience | Asynchronous | Decentralized |
|---|---|---|---|---|---|
| **Distance-based GARs** | | | | | |
| **FedAvg** [87] | N/A | N/A | ✓ | ✗ | ✗ |
| **CM** [55] | $M \geq 2b + 1$ | ✓ | ✗ | ✗ | ✗ |
| **CTM** [55] | $M \geq 2b + 1$ | ✓ | ✗ | ✗ | ✗ |
| **GeoMed [33]** | $M \geq 2b + 1$ | ✓ | ✗ | ✗ | ✗ |
| **Krum** [126] | $M \geq 2b + 3$ | ✗ | ✗ | ✗ | ✗ |
| **Multi-Krum** [126] | $M \geq 2b + m + 2$ | ✗ | ✗ | ✗ | ✗ |
| **Bulyan** [31] | $M \geq 4b + 3$ | ✗ | ✗ | ✗ | ✗ |
| **RSA [173]** | $M \geq 2b + 1$ | ✓ | ✓ | ✗ | ✗ |
| **SignSGD [35]** | $M \geq 2b + 1$ | ✓ | ✗ | ✗ | ✗ |
| **ByRDiE [60]** | $M \geq 2b + 1$ | ✗ | ✗ | ✗ | ✓ |
| **BRIDGE [61]** | $M \geq 2b + 1$ | ✗ | ✗ | ✗ | ✓ |
| **Performance-based GARs** | | | | | |
| **RONI [174]** | $M \geq b + 1$ | ✓ | ✗ | ✗ | ✗ |
| **TRIM [137]** | $M \geq b + 1$ | ✓ | ✗ | ✗ | ✗ |
| **Zeno [41]** | $M \geq b + 1$ | ✗ | ✗ | ✗ | ✗ |
| **Zeno++ [42]** | $M \geq b + 1$ | ✗ | ✗ | ✓ | |
| **PDGAN [65]** | $M \geq b + 1$ | ✓ | ✗ | ✗ | ✗ |
| **MOZI [160]** | $M \geq b + 1$ | ✓ | ✗ | ✗ | ✓ |
| **Other GARs** | | | | | |
| **FoolsGold** [16] | N/A | ✓ | ✓ | ✗ | ✗ |
| **DRACO [34]** | $M \geq 2b + 1$ | ✓ | ✗ | ✗ | ✗ |

| | | | | | |
|---|---|---|---|---|---|
| **Pro-Bristle** | $M \geq b + 1$ | ✓ | ✓ | ✓ | ✓ |

**TABLE 1.** OVERVIEW OF THE MOST NOTABLE GARS

## 4.2. Our solution

To provide Byzantine-resilience, we first observe that in a perfect world (non-i.i.d., all peers working synchronously on the same iteration, and with a balanced dataset) for every node $A$, all benign models that node $A$ receives will be reasonably close to node $A$'s own model. However, Byzantine models that the node receives can be either within or outside this distance. This inspires us to first apply a **distance-based filter** to get rid of some Byzantine attacks, and then apply a **per class performance-based integrator** to integrate the resulting models and filter out more sophisticated Byzantine attacks. The distance-based filter simply takes the $n$ closest models (Euclidean distance), but also supplements these models with a few random samples of the non-selected models (see *exploration vs exploitation strategy* in Section 5.2) in case a few models were received that are so much better than the current model that they end up having a high distance to the peer's own model. The distance-based filter also pads the resulting list of models with previously received models when the number of models received is too small (see *model buffer* in Section 5.2) to make the filter more robust. The resulting models are then evaluated with a test dataset by the performance-based integrator. Because the performance-based integrator also plays a major role in achieving non-i.i.d.-resilience, we will explain that component in more detail in Section 6.2.

# 5. Achieving asynchrony-resilience

## 5.1. Related research

As mentioned in Section 1.2, numerous asynchronous approaches have been proposed [24-28, 115, 116, 130, 131] to solve the problems related to device heterogeneity with synchronous approaches. Asynchronous methods either overprovision clients and then accept the first $x$ updates, dynamically update the synchronization time or amount of work per node, or use weighted averaging based on the staleness of incoming model updates. However, these approaches yield several problems, especially when they are used together with the most common type of Byzantine-resilient defense mechanisms, namely distance-based GARs (see Section 4.1):

- **Stale model**: the model received from another node is stale (outdated, trained on less training data than the other models).
- **Received too few models**: when a node receives a model from another node, it is usually (directly or indirectly) compared against models received from other nodes. In an asynchronous environment, it may happen that only a small number of other models is received in a reasonable amount of time, slowing down the integration of the model.
- **Received a "too good" model**: when a node receives a model from another node that is extremely accurate compared to the other models, it may also be quite different, resulting in an incorrect rejection by a distance-based filter.

Apart from these three problems, a common reoccurrence is that all of these approaches are dependent on a centralized parameter server, while in this thesis we will use a completely decentralized network (see Section 0). Fortunately, truly decentralized federated learning systems such as [29, 153, 213] are in practice always asynchronous because there is no server to synchronize training rounds, but these papers assume that the maximum staleness of peers is bounded. [214] aims to achieve byzantine consensus in decentralized asynchronous networks, but they do not consider a situation where nodes can randomly join/exit the network. [52] presents FedProx, a modification of the FedAvg algorithm that is supposed to tackle heterogeneity in FL by considering a variation of computational power and other differences between devices. However, its performance turns out to be sub-par in later research. Another method based on FedAvg is SAFA [130], which aims to harness the potential efficiency gains of an asynchronous setting while using (a) a pace steering mechanism to reduce the impact of stale models and straggling clients, and (b) an aggregation algorithm that exploits a cache structure to reduce communication costs.

An approach that outperforms the former ones is presented in [53] that performs layer-wise matching and averaging of channels/neurons. It sends at the start of each training rounds global model matching results to the clients and adds additional neurons to the local models to achieve better performance. To prevent the global model from drifting too much towards the fastest nodes, [54] proposes a mechanism to reduces this impact.

## 5.2. Our solution

An asynchronous system can be faster than a synchronous system, especially when the computation power or bandwidth differs significantly between nodes. A faster node does not have to wait for the synchronization step and a slower node does not lose its progression after the next synchronization step after all. To account for an asynchronous environment, we will first differentiate between three sub-problems:

1. *Too few peer models received to reliably perform distance-based screening.* After an iteration, the number of models received from other nodes is arbitrary. When iterations are computed fast (for example, because the node has significant computational resources) and models are received only slowly (for example, because the available bandwidth is limited), the number of received models might be small. When, say, only two models are received, trivial distance-based screening procedures obviously do not work because there are insufficient received models to compare with each other.
2. *Stale model received.* A model that is received might be outdated and stale (trained fewer iterations than the current model). When this model is integrated into the peer's current model, the performance may degrade instead of increase.
3. *Extremely accurate model received that is so different from the peer's own model that it is filtered out by the distance-based screening method.* A model that is received might be way better than the current model. When this model is merged into the peer's current model, the performance will increase significantly. However,

when the most popular type of Byzantine-resilient GARs is used (namely distance-based screening, see Section 4.1) the model might be filtered out because it may differ significantly from the current model to achieve this level of performance.

Unfortunately, the three problems are quite different and therefore we use three separate solutions to solve each of them.

1. *Too few peer models received to reliably perform distance-based screening.* We propose to add a **model buffer** to keep track of a fixed number of recently received models to make the distance-based screening procedure more robust. Applying this idea in a federated setting is not entirely new because it was also explored by Yang et al. [27]. However, the paper of Yang et al. (a) assumes a centralized instead of a decentralized setting, (b) does not explicate what advantage using multiple buffers exactly gives in their solution (in fact, it is entirely unclear), and (c) is unable to update its model directly after receiving an update, resulting in subsequent local training on a (slightly) outdated model.
2. *Stale model received.* To filter out stale models, we first use a fast **distance-based filter**. When the stale model is not stale enough to be filtered out by the distance-based filter, it continues to the accurate per-class performance-based filter. The **performance-based filter** tests the accuracy of every model on a small, trusted dataset. When the performance of a model is subpar, its weight in the weighted averaging of all received models will be very low (or even zero). These two filters are described in more detail in Section 0.
3. *Extremely accurate model received that is so different from the peer's own model that it is filtered out by the distance-based screening method.* To solve this, we propose to use a popular strategy that has to the best of the authors' knowledge never been applied in this context before, namely **exploration vs exploitation**. Based on an exploration ratio $\alpha$, the distance-based screening filter should randomly accept models that are "not close enough" to be considered otherwise. The performance-based filter will then notice the supposedly superior performance of this model and assign the model a high weight for the weighted averaging step (see Section 0).

# 6. Achieving non-i.i.d.-resilience

## 6.1. Related research

Whereas in regular distributed learning environments, a characteristic of a typical federated learning environment is that the shards of the nodes are non-i.i.d. (not independent and identically distributed) [55, 126, 127]. For example, it is possible that device A has class X and Y, and device B has class Y and Z. As a result, the model of device B will be quite different from the model of device A, making it hard for device A to determine whether device B's model is malicious or not. To make matters worse, even though McMahan et al. [87] 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 [30, 49, 87]: the model performs significantly worse than when it would have been trained by a single node on class X, Y, and Z.

The challenge of building a single global model by combining multiple local models without reducing their accuracy is closely related to *Lifelong Learning* (LL), often called C*ontinual Learning* (CL) in the deep learning community *[215].* Continual Learning is concerned with preventing *Catastrophic Forgetting* or *Catastrophic Inference*, a phenomenon where the 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 4 is a Venn diagram created by Lesort and Lomonaco [22] 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 [216-219], 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 [220], 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 [221], and eventually to CWR* by replacing batch normalization with batch renormalization and weight constraining by learning rate modulation [222].

**Regularization** techniques minimize the extent to which the most important weights are overwritten by training on a new model. Elastic Weight Consolidation (EWC) [219], which was based on Learning without Forgetting (LwF) [223] 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 place the importance of the parameters on the diagonal of the *Fisher information matrix*, which works well for learning permutations of the same task, but not for learning entirely new categories incrementally [224]. Several improvements have been made since such as [225-228].

**Rehearsing** old samples interleaved with new samples is also an effective way to prevent catastrophic forgetting. [30] 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 [229-232].

**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.

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. [51] 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

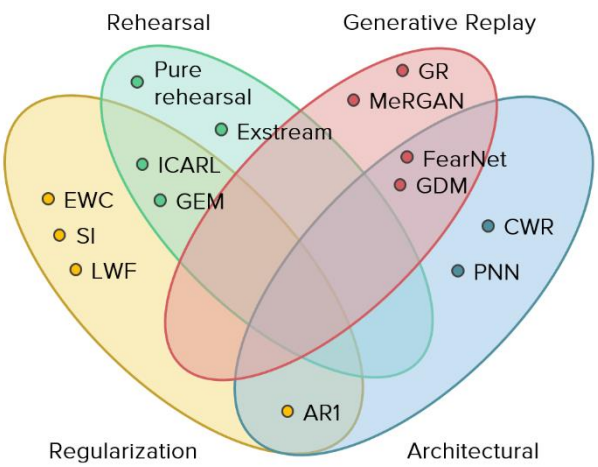



**FIGURE 4. VENN DIAGRAM OF EXISTING CL METHODS [2-5, 7, 9-11, 16, 19-23]**

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. [30] 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 (which is generally a realistic assumption).

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 [30, 233-235]. Mocha [235] handles non-i.i.d. datasets well while also tackling 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, like many other CL algorithms, it assumes that all peers participate in each training round which makes these algorithms harder to apply in a truly federated setting.

A particularly popular approach to use in federated learning systems seems to be Elastic Weight Consolidation ([50, 236-239]) 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 [240].

Another way of handling data heterogeneity is by combining the models of nodes with similar data distributions. For example, Bellet et al. [241] and Vanhaesebrouck et al. [242] 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.

In our solution, we will use CWR* as explained in Section 6.2. Unfortunately, there is a small error in the paper on CWR* [222]: 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 [243]. We also think that *working memory* and *consolidated memory* are more accurate terms than short-term and long-term memory.

## 6.2. Our solution

To make Pro-Bristle able to perform well in non-i.i.d. environments, we will investigate **CWR*** as it showed excellent performance in non-federated environments [240] and has not yet been used in federated environments before. As mandated by CWR*, each node uses the same high-quality frozen layers except for the output layer. To update the output layer, all models received from other nodes are evaluated with a private, small, trusted dataset and integrated with a carefully crafted Sigmoid weighted averaging function.

The first challenge that we address is that CWR* requires all nodes to agree on the same high-quality set of non-output layers. In a typical federated learning scenario this is impossible because the whole purpose of federated learning is to actually learn all layers. However, we can solve this problem by making an additional assumption: we assume that for the dataset we want to learn, there exists another publicly available dataset with roughly the same *features* (for example, English words in the case of a language-based dataset, or pixel patterns in the case of an image-based dataset). In many cases, this assumption is realistic considering the rapidly increasing popularity of transfer learning [244]. Assuming that this assumption applies to the dataset being learned, we can use **transfer learning** to learn the non-output layers offline and then transmit these layers to all nodes where they are frozen for use by CWR*. A (major) additional advantage of using transfer learning that we significantly decrease the bandwidth requirements since only the relatively small output layer is shared across the network (see Section 7.2).

The second challenge that we address is obtaining a reliable "certainty" that a received model is benign (this certainty is an essential element of the integration, see Figure 5). When a peer has lots of samples of each class, it is easy to calculate a certainty score since the peer can simply check the recall of each class (an equivalent term for accuracy per class) of each received model on its own dataset. However, when the data is highly non-i.i.d. where a peer has samples of only a tiny part of all classes, this approach clearly does not work because the datasets of two peers may overlap only partially or not at all. In that case, when we assume it is infeasible for an attacker to learn the peer's data distribution, we can get a much better estimate of whether the other peer is benign when we know the overlap between the datasets.

However, the class distribution of other peers is usually unknown in a federated setting since the training data of every node is private (and it would seriously harm the user's privacy when this information would be exposed). To solve this problem, we slightly change CWR*'s behavior and generify the binary does-own-class/does-not-own-class problem to a continuous scale of the extent to which we think that integrating a particular class improves the model, then we check the intersection cardinality $\omega$ between the datasets of the two peers communicating to each other (by using PSI-CA, explained later in this section), and finally we select from all classes that the peer can evaluate the $\omega$ classes with the highest recall for each peer. Based on these classes, we can get a better estimate of the certainty that the other peer is benign. Later in this section, we will elaborate on how this value is calculated and how it is used.

Every iteration, the recall of every class of both the peer's own model and all received models is tested on a private, small, trusted dataset. Subsequently, these recall values are used to update the model's parameters. Note that we measure the recall of each class of which our node has a sufficient number of samples instead of measuring the accuracy of the entire model. This may seem straight-forward but is actually quite different from most literature (for example, Mozi, RONI and TRIM - see Section 4.1 - measure the overall accuracy). Also note that the performance-based integration depends on the availability of sufficient private data samples (for example, data samples from the peer itself). This means that Pro-Bristle ignores models from peers completely until enough private data samples are available to test other models with the desired confidence level.

To explain how the *recall* per class (an equivalent term for accuracy per class) is used to integrate the parameters of incoming models, we show the algorithm in Figure 5 along with a hypothetical example. Let us suppose that the class overlap between our peer and another peer is 3 (as measured by PSI-CA, explained later in this section), and that the recall of the three best performing classes of the other peer is $< 1.0, 0.7, 0.6 >$ and the recall of the corresponding classes of our peer is $< 0.7, 1.0, 0.0 >$. The certainty is then calculated by subtracting the standard deviation from the average recall of these 3 values of the other peer's model. This certainty is a (rough) estimate of the degree to which the other peer is benign, rewarding a high average recall and punishing a high standard deviation. In the next step, we differentiate between the three classes previously selected (called *familiar classes*) and all other classes (called *foreign classes*).

On the left side of the diagram, we look at the familiar classes and start by calculating the difference between the recall per class of the peer and the other peer multiplied by a certain constant (influencing how aggressive good or bad scores are penalized). To prevent a sybil attack where a large group of sybils slowly degrade the model's performance, an attack penalty $pen_{attack,i}$ is applied that penalizes subsequent performance degradations per class. Subsequently, for every familiar class the score is calculated dependent on whether the model from the other peers performs better or worse than the peer's own model. Performance degradations are penalized significantly more than that excellent scores are boosted, because it is significantly easier to degrade a model's performance than to improve it. Finally, the score per familiar class is fed into a sigmoid function, linearly scaled to the range $[0, 6]$, and multiplied by the certainty score that we calculated earlier.

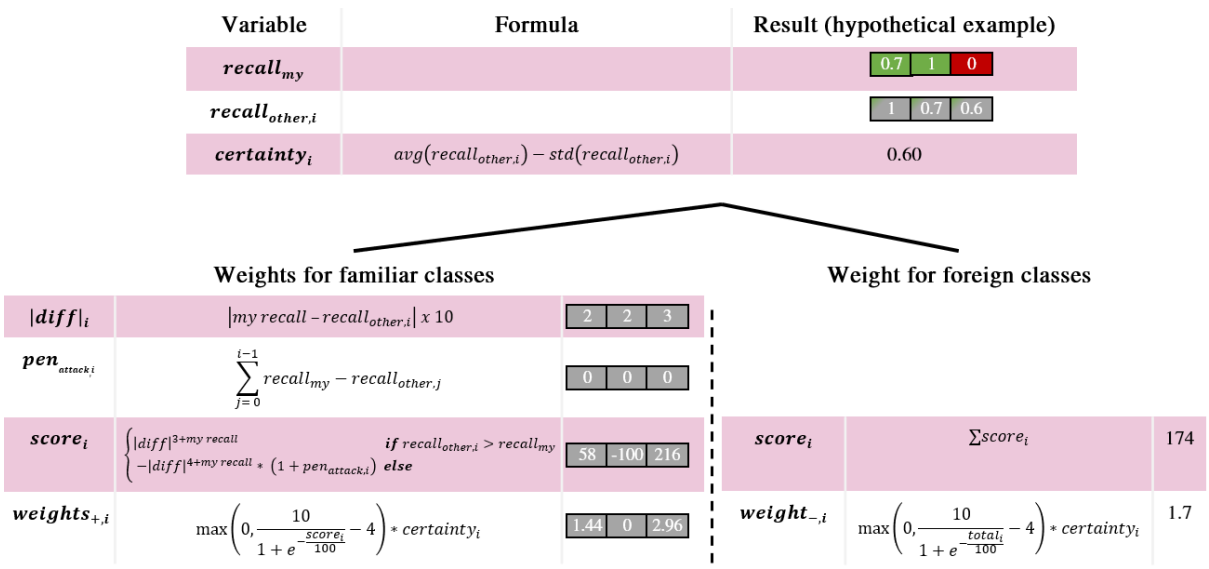



**FIGURE 5.** INTEGRATION OF PARAMETERS

The right side of the diagram shows how the weight is calculated that is applied to the foreign classes, i.e., the classes for which our peer cannot (reliably) check their performance. This is done by summing up the score of all familiar classes and then using the same sigmoid function as for the familiar classes to obtain the weight for all foreign classes.

In the tests we assume that the network is strongly connected for evaluation purposes (as mentioned in Section 9.8), but in practice Pro-Bristle also works fine when the network is cut into separate disconnected segments. For example, in a hypothetical situation where half of the nodes (i.e., set $A$) only have classes 1, 2, and 3, and where the other half of the nodes (i.e., set $B$) only have classes 4, 5, and 6, the nodes in sets $A$ and $B$ will simply learn a different model. Once a few peers from $A$ or $B$ obtain a sufficiently high number of samples of the other set, the model will slowly converge to a single model shared by all nodes in both $A$ and $B$.

There are a few choices that the developer has to make to use this algorithm.

- The minimal class overlap between two peers. The higher this value, the harder it is for other peers to estimate the peer's class distribution and the more reliable the peer can calculate the certainty score for other peers. The downside of a higher value is that the number of peers with enough overlap may become (too) low to achieve satisfactory machine learning performance.
- The minimum number of samples per class that is sufficient to check the recall per class per model. The higher this number, the more reliable the peer can determine the recall, but the fewer classes for which sufficient samples are available to evaluate.
- 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 also formerly foreign classes, but also the higher the impact when a potentially malicious model is integrated.
- The boost for above-average performing models and 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 extremely bad on foreign classes.

### Private Set Intersection Cardinality (PSI-CA)

Pro-Bristle uses estimates of the peers' class overlap to significantly improve the performance in the performance-screening phase, as well as potentially substantially reduce the number of peers communicating with each other in non-i.i.d. situations (see Section 7.2). This class overlap is approximated by first using Private Set Intersection Cardinality (PSI-CA) to determine the class overlap size $\omega$, and subsequently take the $\omega$ best-performing classes. Note that when the recall per class varies a lot or when some foreign classes happen to be classified very accurately, this method will not yield an accurate estimation of the class overlap. However, this is not a problem for the integration itself, as integrating the best parameters is more important than integrating the most recently trained parameters.

The PSI-CA is implemented similarly as in the work of Lv et al. [245], but whereas Lv et al. used Pohlig-Hellman exponential cipher [246] as commutative encryption method which is computationally expensive, we will use the more efficient SRA as proposed by Shamir, Rivest and Adleman [247] and later rediscovered (or not properly referenced) by Cristofaro et al. [248]. Although one peer cannot directly determine the exact classes that another peer owns, it is easy to obtain this information indirectly by simply submitting an exhaustive set of PSI-CA requests [249]. To avoid this, we require the peers to submit at least, for example, four 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 (for example, 1 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 always determine the classes owned by some particular peer due to the nature of the class-overlap problem, but these measures make it significantly more difficult to do so for a large number of nodes. The PSI-CA protocol used by Pro-Bristle is visualized in Figure 6. Suppose that two peers both have set of classes that they do not want to show to the other peer but do want to know the cardinality of the overlap between these sets. When the nodes are created, they create a *commutative encryption* keypair (commutative encryption means that $Enc_A(Enc_B(x)) = Enc_B(Enc_A(x))$ ). Subsequently, they first encrypt each class, shuffle these encrypted values, and send them to the other peer. The other peer also shuffles these values and re-encrypts them by using its own secret. The other peer also encrypts its own classes similarly to peer 1 and puts them in a bloom filter to reduce the bandwidth needed. Thereafter, peer 2 sends both encryptions back to peer 1 which is able to decrypt the first one (thanks to the commutative

encryption) and compare it to the entries in the bloom filter. Finally, the cardinality of the private set intersection is given by simply counting the number of entries that overlap in the bloom filter.

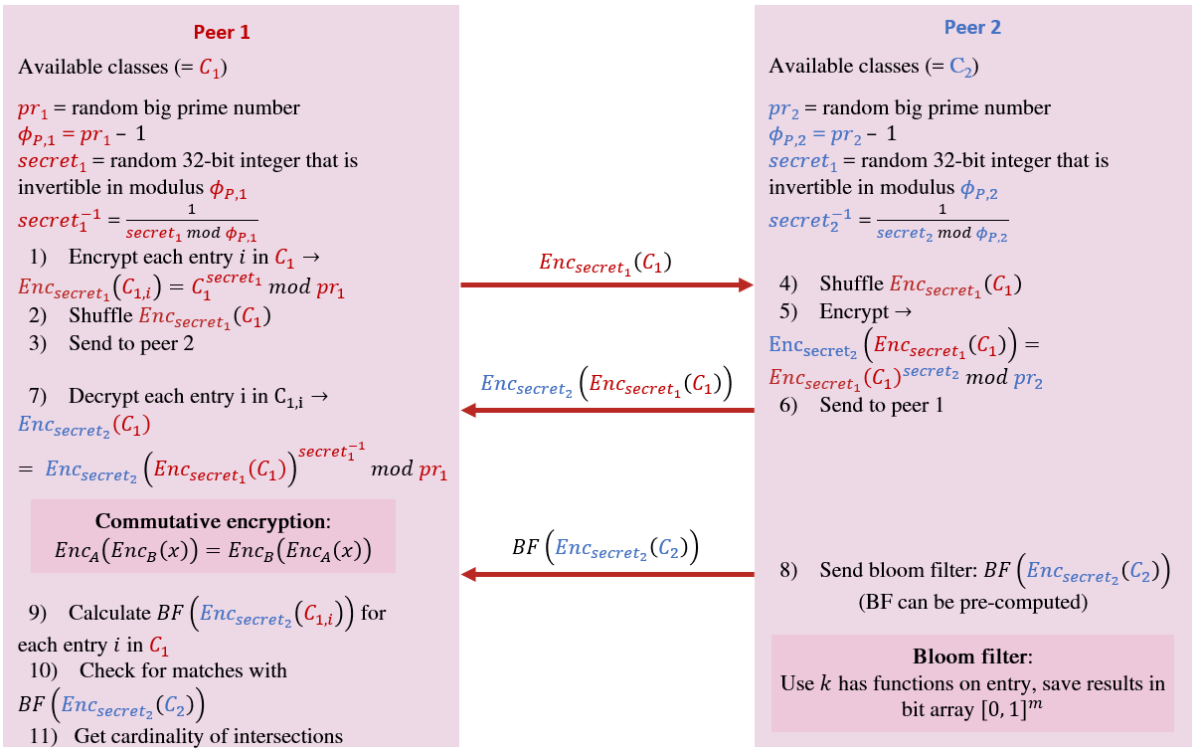



**FIGURE 6.** VISUALIZATION OF SRA PRIVATE SET INTERSECTION CARDINALITY (SRA PSI-CA)

# 7. Achieving communication-efficiency

## 7.1. Related research

Because federated learning can consume a considerable amount of bandwidth, it can create a bottleneck and lead to high network communication costs. The following formula bounds the number of bits that are transmitted by every node during the training [250]:

$$b \in \mathcal{O}\big((H(\Delta\mathcal{W}) + \eta) * |\mathcal{W}| * N_{iter} * f\big)$$

where $H(\Delta\mathcal{W})$ is the entropy of the model's parameters, $\eta$ 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)[251].

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, and by using probabilistic quantization [252] in which the gradients are quantized to low-precision values. The latter option has been researched extensively. Whereas SignSGD and its variants ([35, 253]) quantize the gradient parameters to a single, Feldman et al. [254] generalized the algorithm to a low-bit SGD version. Similar efforts were made by Alistarh et al. [255], Wen et al. [256], and Zhou et al. [257].

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 [255, 258]. Strom [259] noted that most parameters of a neural network are close to zero and therefore suggested only transmitting gradients greater than a predetermined constant threshold (later improved by [260]). Because this threshold is hard to determine, as 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. [261] and Aji and Heafield [262] select a fixed proportion of the parameters to be transmitted, Chen et al. [263] adjust the compression rate dynamically based on local gradient activity, and Tao et al. [264] 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, [265] manages 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, [266] only transmits the model if the accuracy of the updated model is sufficiently higher than the former model. Tang et al. [120] introduced 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. [267] proposes ADSP that lets nodes transmit their model at strategically decided intervals, and Wang et al. [268] optimizes the transmission frequency based on the resource budgets between all participating nodes.

## 7.2. Our solution

In contrast to most literature, we do not focus on compressing the transmitted gradients as much as possible, but rather on limiting the number of peers that a model is shared with and the number of layers that are transmitted. It is certainly possible to combine existing gradient compression methods with our approach, but we believe that our approach naturally blends in with the other components of Pro-Bristle while gradient compression methods are comparatively unrelated to Pro-Bristle.

We use three methods to communicate more efficiently:

- The first method is to only send the model to peers that are able to check the model's performance and thus reliably integrate the transmitted parameters into their model. This is done by using Private Set Intersection-

Cardinality (PSI-CA), as explained in detail in Section 6.2. To put it in PSI-CA terms, nodes only communicate with each other when the cardinality between the intersection of their private datasets is sufficiently large.

- The second method is to transmit only the parameters of the output layer, which has a very significant impact on the amount of data transmitted. This is possible by using a mixture between deep transfer learning and CWR*, as explained in Section 6.2 as well.
- The third method is straightforward, namely a gzip compression of the data stream. Gzipping the transmitted model (a float matrix) reduces the length of the data stream by 5% to 10%.

In Section 10 we evaluate the impact on the model's accuracy based on the communication pattern used. Whereas in most literature the systems are designed such that each peer communicates with every other peer, we also examine what happens when each peer communicates with a random other peer, when a round-robin communication pattern is used, or when a ring architecture with quadratic steps is used for the communication. The latter method is less common and deserves some extra explanation: Figure 7 shows a visualization of the nodes to which a particular node transmits its model. The step size increases at every step with an increment that doubles at every step (a variation is to use $2^0$, $2^1$, $2^2$, $2^3$, $2^4$, …). This communication pattern ensures that the number of steps required to communicate an update between two nodes is logarithmic in the number of nodes. This contrasts with round-robin, where the number of iterations to communicate an update between two nodes can be the total number of nodes minus 1.

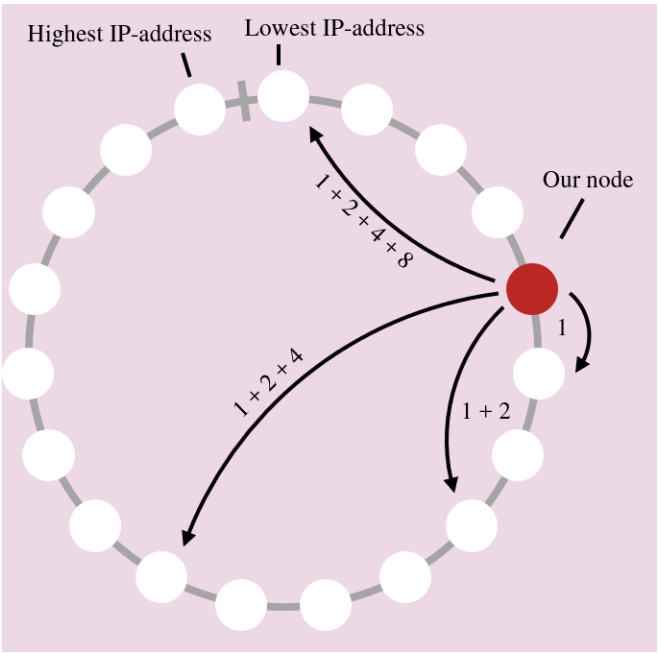



**FIGURE 7.** VISUAL ILLUSTRATION OF RING COMMUNICATION

## 8. Bringing all the pieces together: a high-level overview of Pro-BrisIte

In the introduction of this thesis, we mapped in Figure 1 the challenges that we want to address to the corresponding solutions that we devised. In the subsequent sections, we discussed the related research for each challenge and explained our solution. In this section, we will bring all the separate elements together to form Pro-Bristle. We do this by providing a visual overview in Figure 8 and the corresponding formal pseudocode in Section 8.1. The colored numbers in Figure 8 correspond to the line numbers in the pseudocode.

We chose to use a hypothetical example in Figure 8 to illustrate the integration process of Pro-Bristle. In this example, we have a peer (peer 0) that receives a model from two other peers (peer 1 and peer 2). In this example, peer 0 has a sufficient number of data samples of class 1, 2, 3 and 4; peer 1 has a sufficient number of data samples of class 2, 3, 4, 5 and 8; peer 2 has a sufficient number of data samples of class 6, 7, 8 and 9. With a "sufficient number of data samples" we mean the least number of data samples to achieve a satisfactory degree of confidence in the evaluation of the accuracy of the received models. In our implementation we consider 10 samples per class as sufficient.

By using PSI-CA (see Section 6.2), peer 0 learns that its class overlap with peer 1 is 3 and its class overlap with peer 2 is 0. Let us suppose that the peers require at least a class overlap of 3 to communicate with each other (the lower the class overlap, the less reliably the peer can estimate the accuracy of the other peer and integrate its foreign parameters), then peer 0 will communicate only with peer 1.

At each iteration, peer 0 trains its network by using CWR* (see Section 6.2) that updates the prefrontal cortex-alike short-term memory *tw* and cerebral cortex-alike long-term memory *cw*. Suppose that peer 1 transmitted its model to peer 0. In that case, peer 0 will first apply a distance-based filter to peer 1's model after performing the aforementioned training iteration. Peer 1's model is compared with all other models that peer 0 received (and a buffer of previously received models if too few models were received) and if it passes the distance-based filter, it will continue to the next stage in the integration process. If it did not pass the distance-based filter, there is a probability $\alpha$ that it will be selected for the integration stage anyway, just in case the model is so good that it is quite different (high distance) from peer 0's own model.

To integrate peer 1's model into peer 0's own model, peer 0 uses a few samples of his own dataset (that were not used to train its own model) to test peer 1's recall for every class for which peer 0 has a sufficient number of available samples. In this case, peer 0's own model scores <0.9, 0.7, 0.8, 0.5> and peer 1's model scores <0.3, 0.8, 0.4, 0.9>. The PSI-CA between peer 0 and peer 1 was 3, so peer 0 selects the 3 best performing classes of peer 1 and integrates them with a carefully crafted sigmoid weighted averaging function as explained in Section 4.2 and shown in Figure 5, and optionally also integrates the classes for which peer 0 has an insufficient number of samples as foreign class parameters. The result of the integration is stored in the consolidated memory *cw*, transmitted to other peers, and used for the next iteration.

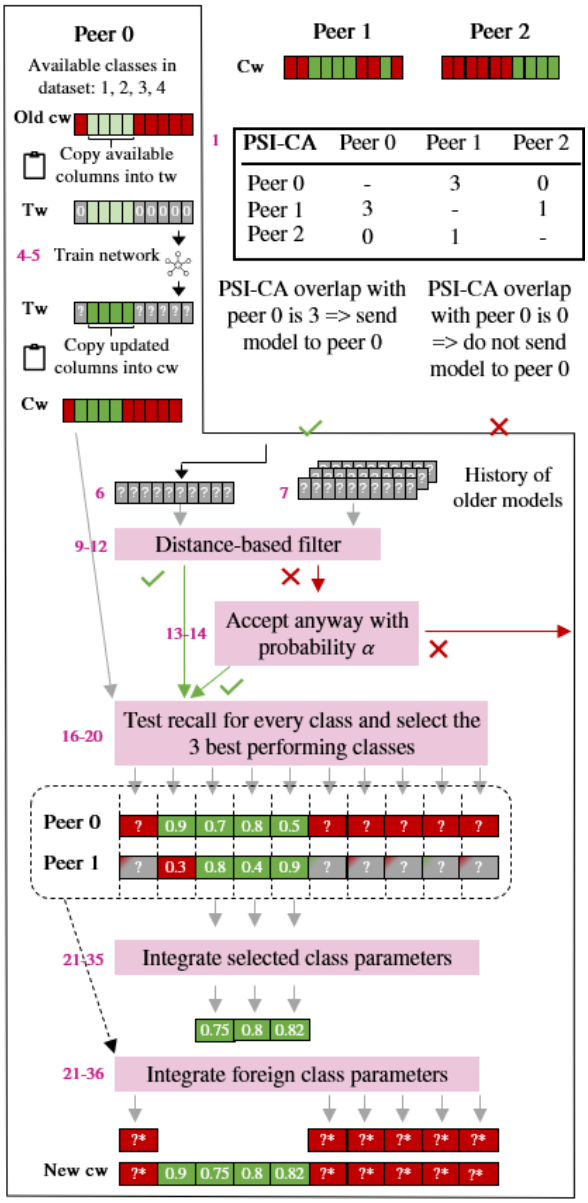



**FIGURE 8. HIGH-LEVEL OVERVIEW OF PRO-BRISTLE. THE COLORED NUMBERS CORRESPOND TO THE LINE NUMBERS IN THE PSEUDOCODE**

## 8.1. Pseudocode

The pseudocode corresponds to (the explanation of) Figure 8, where the colored numbers correspond to the line numbers in the pseudocode.

### Notations

We use bold lower-case letters such as $\boldsymbol{x}$ to represent vectors, lower-case letters such as $\gamma$ to represent scalars and functions, and upper-case curlicue letters such as $\mathcal{N}$ to represent sets. Aggregated vectors are denoted by a line over vectors such as m. All operations between vectors are element-wise operations in this paper (except inner products of vectors).

**Input:** initial estimate $x_0$, dataset $D^{train}$ containing an arbitrary non-validation subset of the node's collected data, dataset $D^{test}$ containing a small, trusted set of samples for each class, history buffer size $\gamma$, exploration ratio $\frac{\beta}{\alpha}$, transfer network $\Psi$, weight diff constant $\eta$

1. $\mathcal{N}^s \leftarrow$ getSimilarPeersWithClassOverlap()
2. $l \leftarrow$ initialize loss function by deep transfer from $\Psi$
3. **For** $t = 0, 1, 2, \ldots$ **do**
4. Stochastically sample $\xi_i(t)$ from $D_i^{train}$
5. $\nabla l(\boldsymbol{x}_i(t), \xi_i(t)) \leftarrow$ Compute the local gradient of output layer
6. $\mathcal{N}_i^n(t) \leftarrow$ All models received from peers $j \in \mathcal{N}^s$
7. $\mathcal{N}_i^r(t) \leftarrow$ The $\gamma$ most recent models received in previous iterations
8. **If** $|\mathcal{N}_i^n(t)| > 0$ **then**
9. **For** $j$ *in* $(\mathcal{N}_i^n(t) \cup \mathcal{N}_i^r(t))$ **do**
10. $d_{i,j} \leftarrow \left|\left|\boldsymbol{x}_i(t) - \boldsymbol{x}_j(t)\right|\right|$
11. **End for**
12. $\mathcal{N}_i^d(t) \leftarrow \left(argmin_{\substack{\mathcal{N}^* \subseteq \mathcal{N}_i \\ |\mathcal{N}^*| = \alpha}} \sum_{j \in \mathcal{N}^*} d_{i,j}\right) \setminus \mathcal{N}_i^r(t)$
13. $\mathcal{N}_i^e(t) \leftarrow \mathcal{N}^* \subseteq \mathcal{N}^*$ *is a random subset where* $|\mathcal{N}^*| = \beta \left(\mathcal{N}_i^n(t) \setminus \mathcal{N}_i^d(t)\right)$
14. $\mathcal{N}_i^c(t) \leftarrow \mathcal{N}_i^d(t) \cup \mathcal{N}_i^e(t)$
15. **For** $j \in i \cup \mathcal{N}_i^c(t)$ **do**
16. **For** $c$ *in* $classes(D_i)$
17. $recall_{j,c}(t) \leftarrow recall(l, \boldsymbol{x}_j(t), D_i^{test})$
18. **End for**
19. **If** $j \neq i$ **then**
20. $C_j(t) \leftarrow argmin_{\substack{c \text{ in } classes(D_i) \\ |C| = cardinality(\mathcal{N}_j^s(t))}} recall_{j,c}(t)$
21. $certainty_j(t) \leftarrow \max\left(0,\ average\left(recall_{j,c \in C_j(t)}(t)\right) - 2 * std\left(recall_{j,c \in C_j(t)}(t)\right)\right)$
22. **For** $c$ *in* $C_j(t)$
23. $weightdiff_{j,c}(t) \leftarrow \left|recall_{j,c}(t) - recall_{i,c}(t)\right| x\ \eta$
24. $seqAttackPenalty_{j,c}(t) \leftarrow getSeqAttackPenalty(C_j(t), c, recall)$
25. **If** $recall_{j,c}(t) > recall_{i,c}(t)$
26. $score_{j,c}(t) \leftarrow weightdiff_{j,c}(t)^{3+recall_{i,c}}$
27. **Else**
28. $score_{j,c}(t) \leftarrow -weightdiff_{j,c}(t)^{4+recall_{i,c}} * \left(1 + seqAttackPenalty_{j,c}(t)\right)$
29. **End if**
30. $weights_{j,c}(t) \leftarrow \max\left(0, \frac{10}{1+e^{-\frac{score_{j,c}(t)}{100}}} - 4\right) * certainty_j(t)$
31. **End for**
32. $peerWeight_j(t) \leftarrow \max\left(0, \frac{10}{1+e^{-\frac{\sum_{c \in C_j(t)} score_{j,c}(t)}{100}}} - 4\right) * certainty_j(t)$
33. **End if**
34. **End for**
35. $\boldsymbol{x}_i(t) \leftarrow weightedAvg(\boldsymbol{x}(t), weights(t), C)$
36. $\boldsymbol{x}_i(t) \leftarrow weightedAvg(\boldsymbol{x}(t), weights(t), all\ classes \setminus classes(D_i))$
37. **End if**
38. $\boldsymbol{x}_i(t+1) \leftarrow \boldsymbol{x}_i(t) - \lambda \nabla l(\boldsymbol{x}_i(t), \xi_i(t))$
39. **End for**

**EQUATION 5** **FORMAL PSEUDOCODE OF PRO-BRISTLE**

# 9. Implementation

In this section, we aim to give the reader a good understanding of how Pro-Bristle was implemented and the experiments were conducted. At the end of the section, we also describe the biggest issues that we encountered during the development of the system.

## 9.1. Datasets

Of all the papers that we read as part of the literature review for this thesis, two datasets turned out to be extremely popular in the literature, namely the **MNIST** dataset[1] and to a lesser extent also the **CIFAR-10** dataset[2]**.**

The MNIST dataset consists of 60,000 gray-scale training images and 10,000 test images of 28x28 px representing handwritten digits. Even though MNIST does not represent a typical federated learning dataset, it is very popular and, thanks to its status as one of the most popular machine learning datasets, it is possible to compare our results with a large body of established literature. To transfer-learn MNIST, we first train the network on the EMNIST-Letters dataset which is similar to MNIST, but contains the 26 letters of our alphabet rather than digits.

The CIFAR-10 dataset also consists of 60,000 training images and 10,000 test images. These images are 32x32 px and RGB-colored, showing pictures of ten distinct types of objects such as cars, airplanes, and dogs. CIFAR-10 turns out to be significantly more challenging to learn than MNIST, which might be useful to properly investigate the power of new algorithms. To transfer-learn CIFAR-10, we first train the network on CIFAR-100. CIFAR-100 is a similar dataset as CIFAR-10 but contains 100 classes rather than 10 classes. Similarly to [269] we reduce the conceptual overlap between CIFAR-10 and CIFAR-100, by excluding super-classes of CIFAR-100 that are conceptually similar to CIFAR-10 classes: vehicle 1, vehicle 2, small mammals, medium-sized mammals, and large carnivores.

We also include a realistic federated learning dataset, namely the **WISDM** dataset, one of the most popular HAR (Human Activity Recognition) datasets [270]. This dataset consists of 1,098,207 recordings of people performing one of the six included activities. Every recording consists of at least 544 measurements of the acceleration sensor. To transfer-learn this dataset we pretrain the network similarly to [271] on the MobiAct dataset where we again exclude overlapping classes with the WISDM dataset.

## 9.2. Machine Learning part

For MNIST and CIFAR-10, we use the same CNN architectures used by McMahan et al. [87] with the only difference that we use Leaky ReLu instead of the regular ReLu as activation function for the hidden layers, since the former one suffers less from the vanishing gradients problem. For the output function we use the softmax function and as loss function, we use negative loglikelihood.

### MNIST

| Layer | #input neurons | #output neurons | Kernel | Stride | Padding |
|---|---|---|---|---|---|
| *Convolution* | 1 | 10 | <5, 5> | < 1, 1> | <0, 0> |
| *Max pooling* | - | - | <2, 2> | <2, 2> | <0, 0> |
| *Convolution* | 10 | 50 | <5, 5> | < 1, 1> | <0, 0> |
| *Max pooling* | - | - | <2, 2> | <2, 2> | <0, 0> |
| *Output* | 800 | 10 | | | |

[1] MNIST was used in the following papers that were cited in this thesis: [4, 6, 7, 9, 12, 14, 16, 18, 21, 23, 26, 30-35, 38, 40, 45-49, 51-78]

[2] CIFAR-10 was used in the following papers that were cited in this thesis: [1, 4, 7, 13, 24-50]

CIFAR

| Layer | #input neurons | #output neurons | Kernel | Stride | Padding |
|---|---|---|---|---|---|
| *Convolution* | 3 | 64 | <3, 3> | <1, 1> | <1, 1> |
| *Batch normalization* | 64 | 64 | | | |
| *Max pooling* | - | - | <2, 2> | <2, 2> | <0, 0> |
| *Convolution* | 64 | 128 | <3, 3> | <1, 1> | <1, 1> |
| *Batch normalization* | 128 | 128 | | | |
| *Max pooling* | - | - | <2, 2> | <2, 2> | <0, 0> |
| *Batch normalization* | 128 | 128 | | | |
| *Output* | 8192 | 10 | | | |

WISDM

| Layer | #input neurons | #output neurons | Kernel | Stride | Padding |
|---|---|---|---|---|---|
| *1D convolution* | 3 | 96 | <15> | <1> | <7> |
| *Batch normalization* | 96 | 96 | | | |
| *1D max pooling* | - | - | <2> | <2> | <0> |
| *1D convolution* | 96 | 128 | <9> | <1> | <4> |
| *Global max pooling* | - | - | | | |
| *Batch normalization* | 128 | 128 | | | |
| *Output* | 128 | 6 | | | |

## 9.3. Gradient Aggregation Rules

To properly compare our proposed solution with existing methods, we implemented five other gradient aggregation rules (described in detail in Section 4.1):

- **FedAvg** [87]. FedAvg is equivalent to simple averaging, researched extensively, and very often used as baseline to compare other GARs against.
- **CM (Coordinate-wise Median)** [55]. Coordinate-wise Median is perhaps the simplest, but already a particularly effective Byzantine-resilient GAR, as shown by [57].
- **Krum** [126]. Krum is an extremely popular GAR that selects the model with the minimal local sum of Euclidean distances.
- **Bridge** [61]. A very recent survey paper [57], published in May 2020, concluded that Bridge was the best performing GAR in decentralized settings.
- **MOZI** [160]. MOZI was published shortly after [57]'s survey and uses a hybrid between distance-based and performance-screening to achieve superior results.

We initialized all GARs that are dependent on a-priori knowledge of the number of attackers (namely Krum and Bridge) with $b = 4$.

## 9.4. Environment

We use two separate ways to test Pro-Bristle's performance, namely in a local simulation and in a truly decentralized environment. In the first case, we run a single program that iteratively trains and combines up to 58 models to simulate a small-scale federated setting. This approach is not only relatively fast, but also makes it easy to accurately control a variety of settings such as low computing power, low bandwidth, nodes joining / leaving randomly, etc. We also emulate 16 completely independent smartphones to test if the results are comparable in a "real" setting. Unfortunately, this limit of 16 emulators is hardcoded in the Android emulator executable, making it unpractical to run more emulators simultaneously. However, 16 emulators are enough to accurately measure the performance of the different GARs and,

if the programs works well on 16 emulators, give us confidence that the code works as intended and will also scale to a higher number of nodes given the gossiping nature of the system.

## 9.5. Network connectivity

We implemented 4 TODO

## 9.6. Network protocol

The nodes that use federated learning to collaboratively learn a model communicate with each other over a network. Since we aim to run everything fully decentralized, it is non-trivial for nodes to find and communicate with each other in a fault-tolerant and effective way. Therefore, we use IPv8 [272, 273], a well-established decentralized peer-to-peer (P2P) middleware stack that is used by i.a. the popular Tribler media sharing system [274, 275]. Furthermore, we extended 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) that enables parallel transmission of multiple files between the same two nodes. This was 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, to speed up the slow transmission times of TFTP, the first Kotlin implementation of the micro-Transport Protocol (µTP). This protocol aims to mitigate the poor latency and congestion control problems found in regular TCP implementations, while providing reliable and ordered packet delivery. It sends multiple packets simultaneously and automatically slows down the transmission when the network seems to get congested.

## 9.7. Task automation

Creating and starting all emulators, and installing, starting, initializing, running, and evaluating the federated learning program on every emulator, is infeasible to do by hand for a large number of emulators. Therefore, we created a separate coordinator program that automates these tasks. Based on the current operating system, it executes several scripts (for example to create new emulators that are reset to factory settings, or to increase the local network buffers to decrease the uncontrolled / unintended packet loss to speed up network communication) to create and run all tests consecutively. The nodes are instructed to perform specific tasks that are stated in a dedicated JSON file and subsequently communicate their evaluations to the coordinator, who writes the evaluations to a CSV file.

A separate Kotlin script was used to collect and process all evaluations, and a Python script was used to generate the figures based on these evaluations as shown in Section 10.

## 9.8. Threat Model

To evaluate the Byzantine-resilience of the GARs listed in Section 9.3, we set up several Byzantine agents that aim to reduce the model's accuracy. We assume that in non-i.i.d. environments, the Byzantine agents do not know which classes the peer under attack has. This assumption is dependent on the cardinality between their datasets, as discussed in detail in Section 6.2. Furthermore, we assume that the nodes under attack have sufficient training data to evaluate and thus consider the integration of peer models (as discussed in Section 4.2), and that the attackers do not use backdoor attacks since these are NP-hard to detect (see Section 2.1). In non-i.i.d. situations we also assume that for each node $x$, the set of all nodes to which node $x$ is (indirectly) connected covers all classes of the entire dataset. Note that Pro-Bristle also works fine when node $x$ and its (indirect) neighbors only have a subset of all possible classes, or even when node $x$ is completely isolated or fully surrounded by Byzantine nodes. The only disadvantage of such a situation is that node $x$ only learns to train the model on the classes that are available and that, when node $x$ would like to predict a new class, its model is not trained to do so. The attacks used are described in detail in Section 2.2. Below, we just list the most important implementation details that are relevant to understand the results.

### Untargeted data poisoning: All-label-flip attack [1]

Every class is mapped to the label of the next class, so when there are 10 classes, then class 0 is mapped to label 1, class 1 is mapped to label 2, …, and class 9 is mapped to label 0.

### Targeted data poisoning: 2-label-flip attack [1]

Class 0 is mapped to label 1 and class 1 is mapped to label 0.

### Untargeted model poisoning: Additive noise attack [23]

We opt for a variant where half of the parameters are set to uniform noise in the interval $[-0.2, 0]$ and the other half of the parameters is set to noise in the interval $[0, 0.2]$.

### Targeted model poisoning: Krum attack [56]

The attack is not executed when there are less than four models (the minimum number of models to execute the attack). We set the attacker's range $b$ to 2, as recommended by the authors.

### Targeted model poisoning: Trimmed Mean attack [56]

Similar to the Krum attack, also this attack is not executed when there are less than four models (the minimum number of models to execute the attack). And again, we set the attacker's range $b$ to 2, as recommended by the authors.

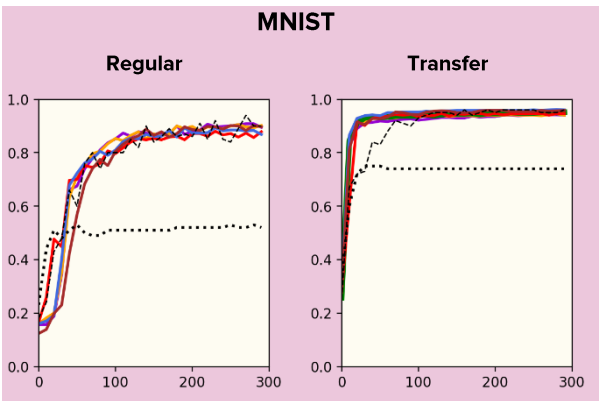
## 10. Results

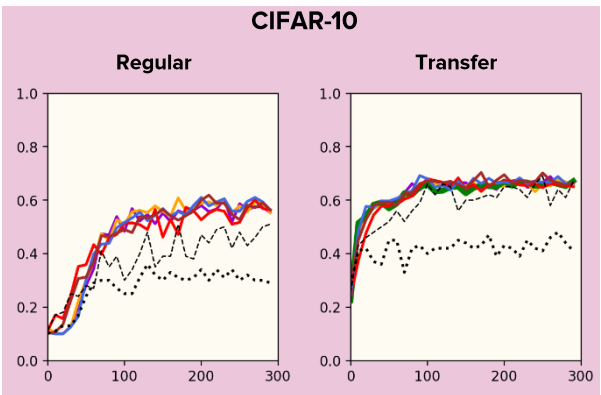
As illustrated in Section 7, Pro-Bristle is hard to compare to existing methods. For instance, it assumes that a large public dataset with roughly similar low-level features is available, which is used for the deep transfer learning to pre-initialize the neural network. As this is an "unfair" advantage with respect to existing GARs that do not rely on a deep transfer stage, for each experiment we first show the performance of existing GARs, and then the performance of existing GARs enhanced by transfer learning and the performance of Pro-Bristle. Additionally, in contrast to almost all existing work on federated learning, we also evaluate the performance when a node simply ignores all received updates. We consider both a **Local Min** scenario where the node has the same tiny amount of data as in the other federated experiments, and a **Local Max** scenario where the node has access to all training data available to all nodes.

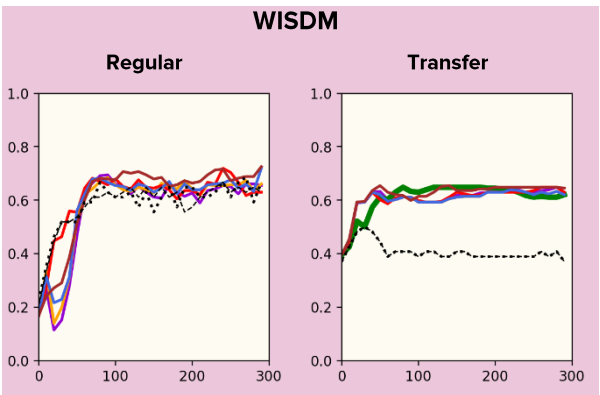
The number of parameters that we can vary in the experiments is obviously extremely large. To keep the number of experiments manageable, we will mostly consider scenarios with 10 benign nodes, with the MNIST dataset, and with all-label-flip attacks. We will first look at the performance of the GARs on the 3 datasets described in Section 9.1 in both a peaceful and a typical Byzantine setting. Subsequently, we evaluate much more Byzantine scenarios where we vary between the type of attack, an i.i.d. vs non-i.i.d. environment, the number of attackers, the communication pattern, and the degree of non-i.i.d.-ness. All results are the average values of all benign nodes.

### Synchronous, Benign, i.i.d. Environment

MLCONFIGURATION *#nodes: 10, architecture, see Section 9.2* DATASET CONFIGURATION ***dataset**: under study, batch size: 5, training data distribution: 10 for each class, test samples per class: 10* NEURAL NETWORK CONFIGURATION *Optimizer: Adam, learning rate: 0.001, momentum: none, l2: 0.005* TRAINING CONFIGURATION ***#iterations**: under study up to 300, **GAR**: under study, **deep transfer**: under study, communication pattern: all, asynchrony mode: none, #iterations before sending: 1, req. class overlap: 2* ATTACK CONFIGURATION *#attackers: 0, attack mode: none*



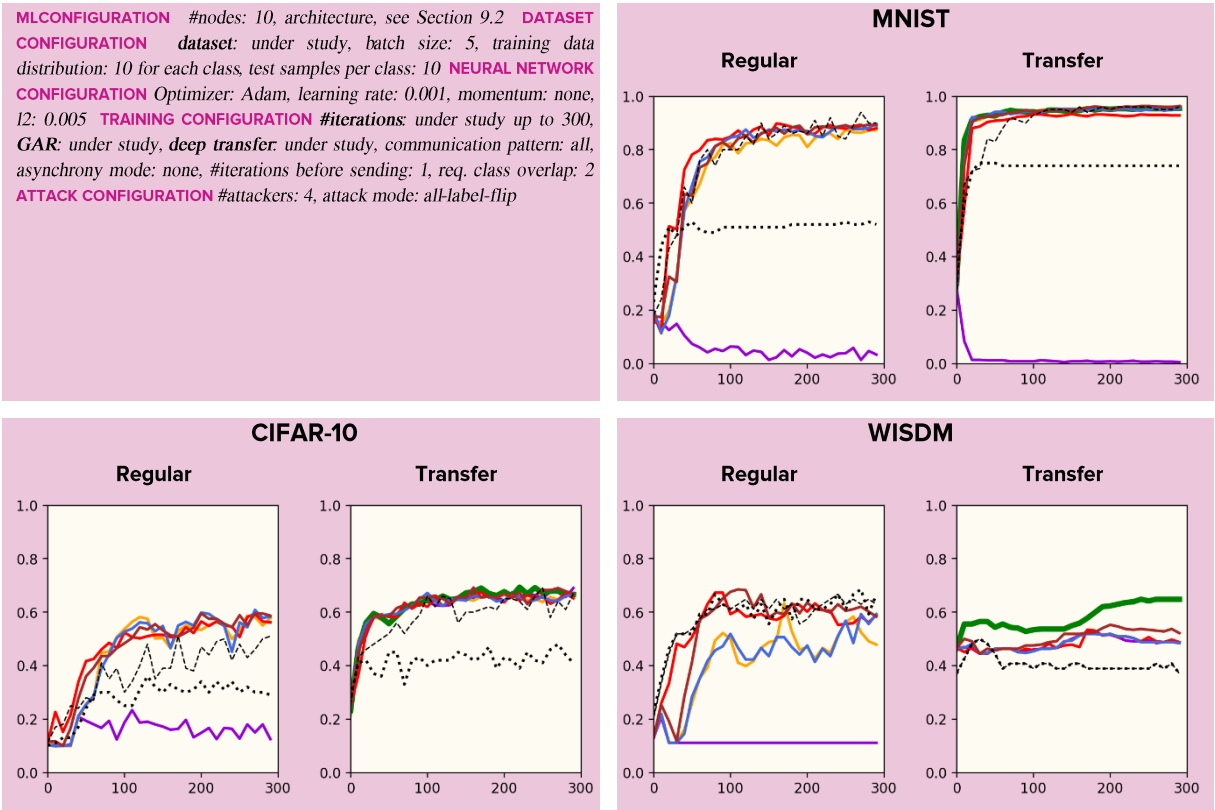




In a "perfect" environment where all nodes communicate in synchronous rounds, are benign, and the data is distributed i.i.d., all GARs seem to benefit from transfer learning and also outperform nodes that ignore updates from their peers. An interesting observation is that the difference between *Local min* and *Local max* is negligible, which indicates that the added value of more training samples is for the first 300 iterations not particularly important for all three datasets. It is also interesting that local training outperforms federated learning on the WISDM dataset. The reason for this behavior is unclear.

## Synchronous, Byzantine All-Label-Flip, i.i.d. Environment

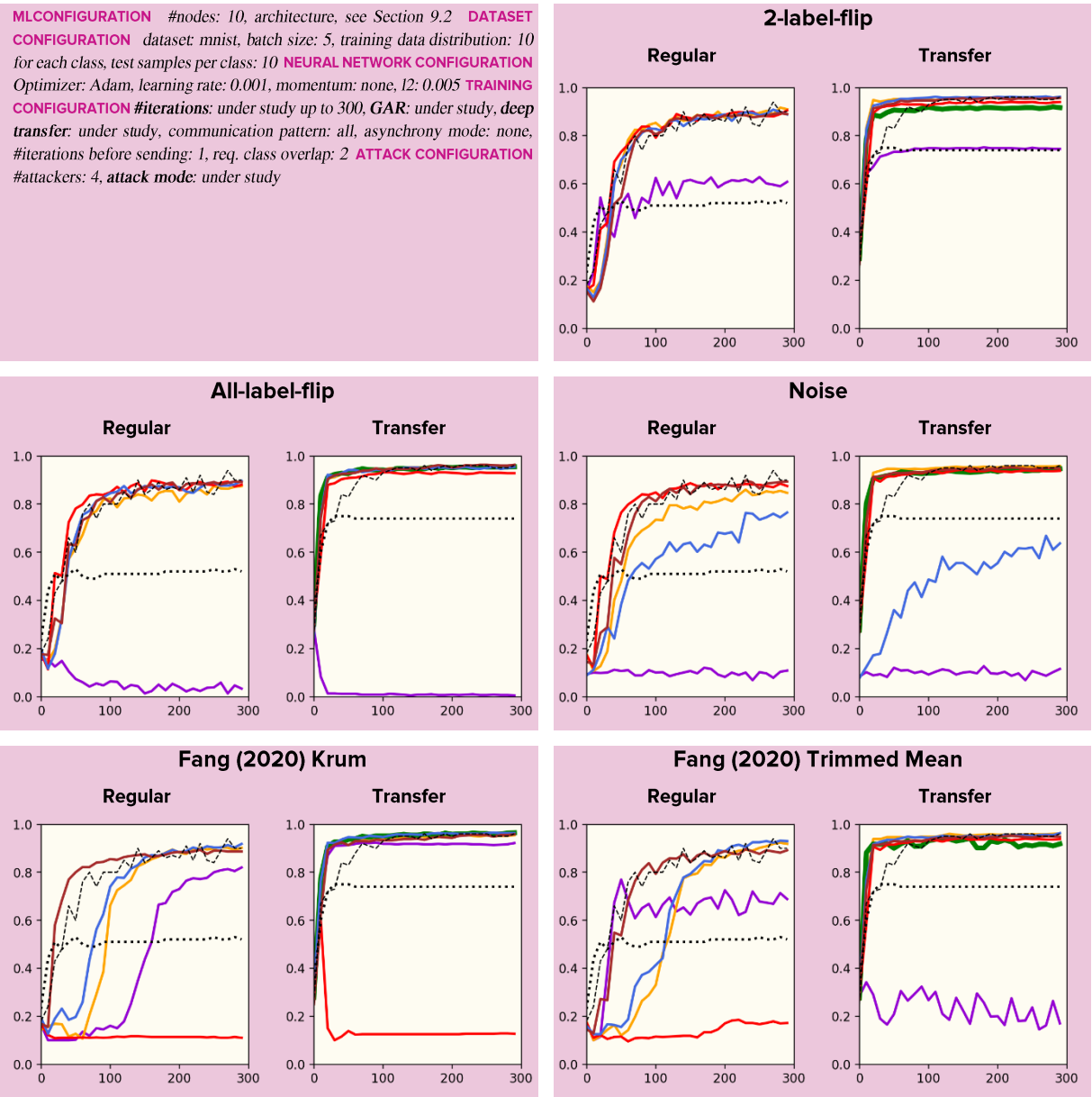
**MLCONFIGURATION** *#nodes: 10, architecture, see Section 9.2* **DATASET CONFIGURATION** ***dataset****: under study, batch size: 5, training data distribution: 10 for each class, test samples per class: 10* **NEURAL NETWORK CONFIGURATION** *Optimizer: Adam, learning rate: 0.001, momentum: none, l2: 0.005* **TRAINING CONFIGURATION** ***#iterations****: under study up to 300,* ***GAR****: under study,* ***deep transfer****: under study, communication pattern: all, asynchrony mode: none, #iterations before sending: 1, req. class overlap: 2* **ATTACK CONFIGURATION** *#attackers: 4, attack mode: all-label-flip*




In a Byzantine environment with a relatively strong all-label-flip attack, it is clear that simple averaging is unable to mitigate the attack in the non-transfer learning situation for each dataset. Interestingly enough, simple averaging is not susceptible to the attack in transfer-learning situation of the CIFAR-10 and WISDM dataset in contrast to the MNIST dataset. The existing GARs that we evaluate seem to do a good job in defending against the attack.

## Synchronous, Byzantine, i.i.d. Environment

**MLCONFIGURATION** *#nodes: 10, architecture, see Section 9.2* **DATASET CONFIGURATION** *dataset: mnist, batch size: 5, training data distribution: 10 for each class, test samples per class: 10* **NEURAL NETWORK CONFIGURATION** *Optimizer: Adam, learning rate: 0.001, momentum: none, l2: 0.005* **TRAINING CONFIGURATION** ***#iterations****: under study up to 300,* ***GAR****: under study,* ***deep transfer****: under study, communication pattern: all, asynchrony mode: none, #iterations before sending: 1, req. class overlap: 2* **ATTACK CONFIGURATION** *#attackers: 4,* ***attack mode****: under study*

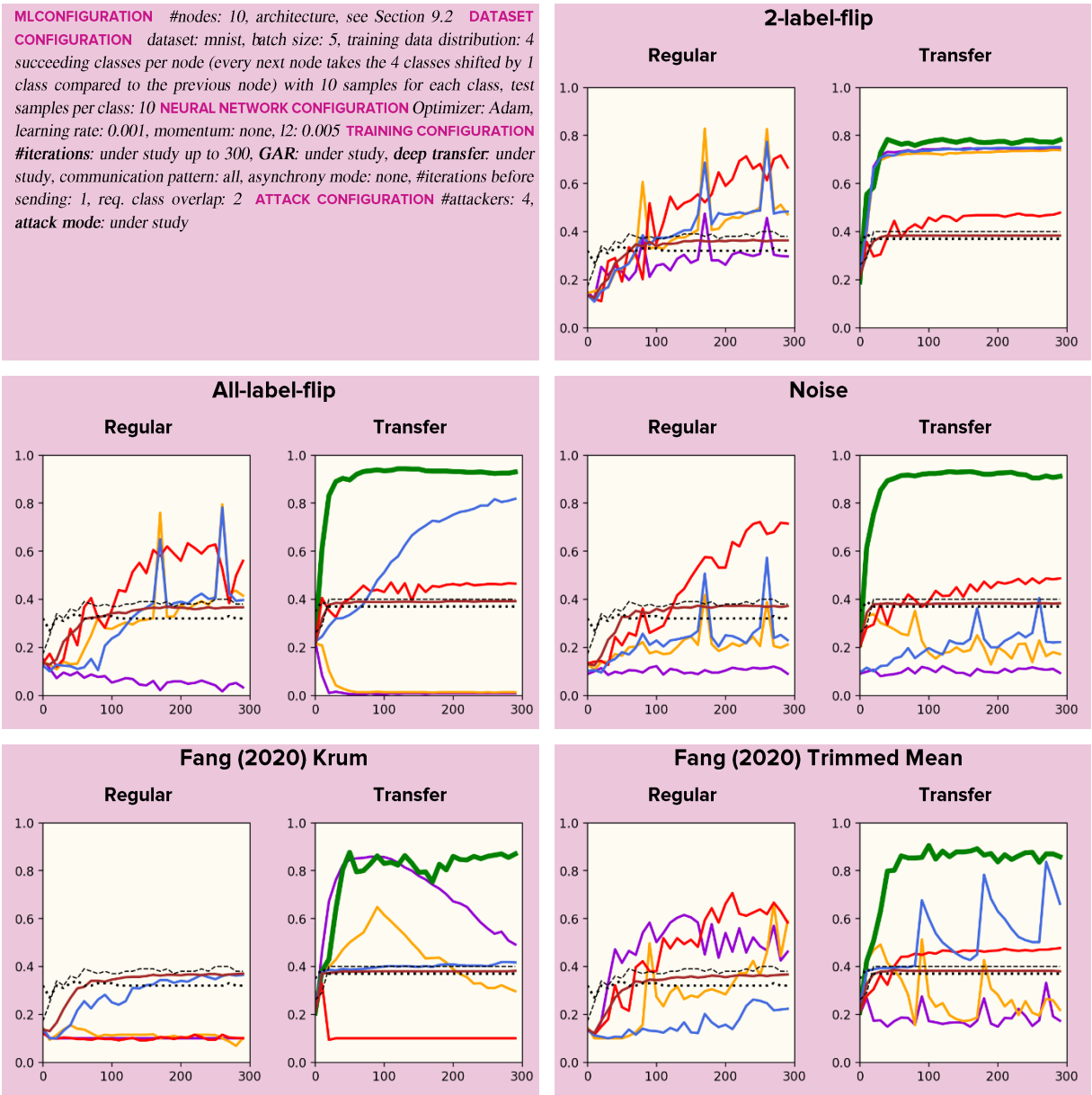



We will further investigate the extent to which the GARs are able to provide protection against Byzantine attacks. A trivial 2-label-flip attack clearly misleads simple averaging but is despite its relatively small influence on the model's parameters successfully mitigated by all GARs. A simple noise attack manages to completely destroy the model when no GAR is used but is again also successfully mitigated by all GARs except the median. The more advanced Fang (2020) Krum attack is clearly very effective against Krum, but only slows down the convergence rate of the other GARs. The Fang (2020) Trimmed Mean is relatively ineffective against any Byzantine-resilient GAR. This is due to the fact that the models of 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.

Average | Bridge | Median | Krum | Mozi | Pro-Bristle | Local min | Local max

## Synchronous, Byzantine, non-i.i.d. Environment

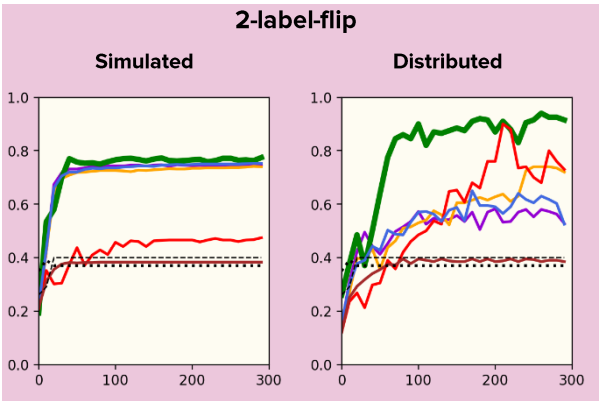
**MLCONFIGURATION** *#nodes: 10, architecture, see Section 9.2* **DATASET CONFIGURATION** *dataset: mnist, batch size: 5, training data distribution: 4 succeeding classes per node (every next node takes the 4 classes shifted by 1 class compared to the previous node) with 10 samples for each class, test samples per class: 10* **NEURAL NETWORK CONFIGURATION** *Optimizer: Adam, learning rate: 0.001, momentum: none, l2: 0.005* **TRAINING CONFIGURATION** ***#iterations****: under study up to 300,* ***GAR****: under study,* ***deep transfer****: under study, communication pattern: all, asynchrony mode: none, #iterations before sending: 1, req. class overlap: 2* **ATTACK CONFIGURATION** *#attackers: 4,* ***attack mode****: under study*

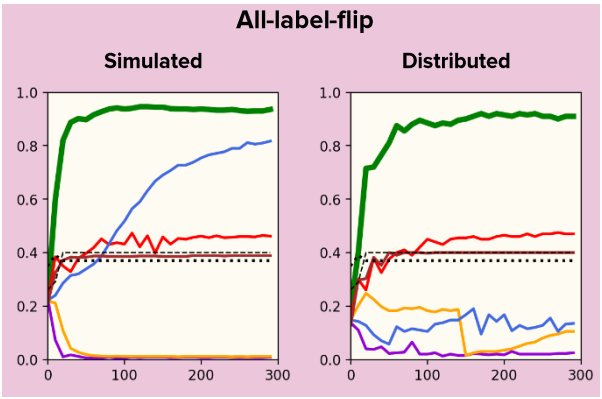


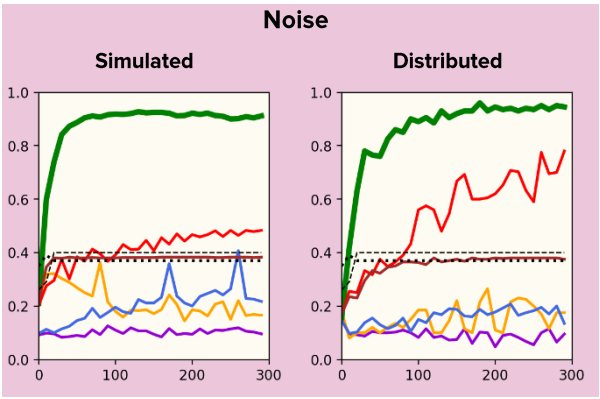

Whereas in the tests so far, Mozi and Pro-Bristle were the only GARs that consistently performed well, this changes in non-i.i.d. situations. In the 2-label-flip scenario, nodes using Krum, Bridge, and Median seem to be able to partly learn about unknown classes from other nodes, but their performance is very shaky. When deep transfer learning is used, the performance becomes quite stable. Although Bridge performed well in the transfer 2-label-flip scenario, it completely breaks down when an all-label-flip attack is used in combination with transfer learning, just like in the noise scenario. Whereas the Fang (2020) Krum attack was quite ineffective against most GARs in i.i.d. environments, it turns out to be extremely powerful in non-i.i.d. environments. Its Trimmed Mean variant crushes the Average GAR in the transfer setting in contrast to the Krum attack, but is only semi-effective against Krum and Median. Mozi consistently defends well against all Byzantine attacks, but also filters out all non-Byzantine vectors, causing the accuracy to be limited to the classes that the peer has locally available. Pro-Bristle consistently outperforms all other GARs, even though it is susceptible to the 2-label-flip attack. This happens because most benign nodes do not have the data of the classes that were flipped by the Byzantine nodes, thus trusting the malicious based on the data that they did have. TODO TE LANG STUKJE TEKST

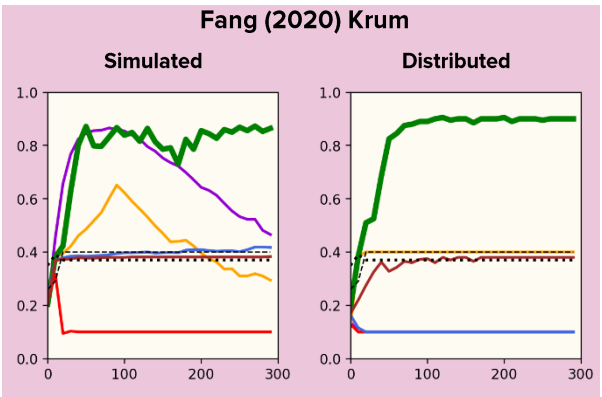
## Synchronous, Byzantine, non-i.i.d. Environment (comparison simulated vs distributed)

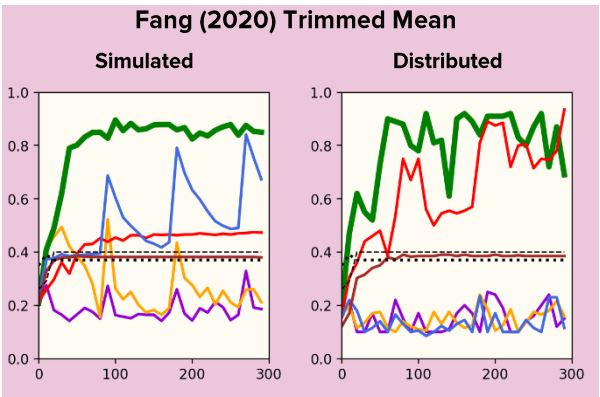
**MLCONFIGURATION** *#nodes: 10, architecture, see Section 9.2* **DATASET CONFIGURATION** *dataset: mnist, batch size: 5, training data distribution: 4 succeeding classes per node (every next node takes the 4 classes shifted by 1 class compared to the previous node) with 10 samples for each class, test samples per class: 10* **NEURAL NETWORK CONFIGURATION** *Optimizer: Adam, learning rate: 0.001, momentum: none, l2: 0.005* **TRAINING CONFIGURATION** ***#iterations****: under study up to 300,* ***GAR****: under study,* ***deep transfer****: under study, communication pattern: all, asynchrony mode: none, #iterations before sending: 1, req. class overlap: 2* **ATTACK CONFIGURATION** *#attackers: 4,* ***attack mode****: under study*



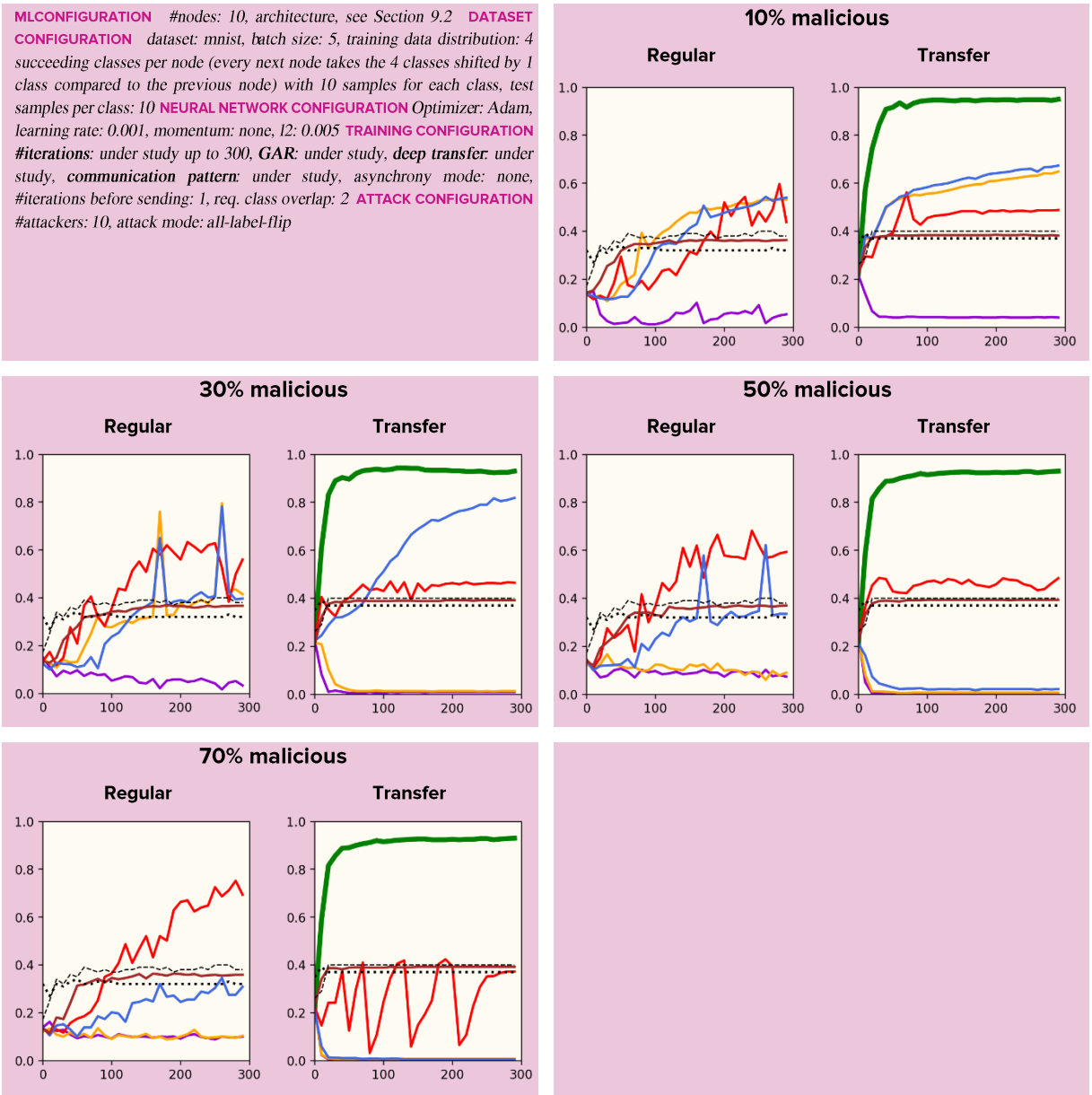




UDP TESTS

TODO WAAROM WIJKEN DE RESULTATEN AF? PRO-BRISLTE WERKT WEL TOP

## Synchronous, Byzantine with varying num. of attackers, non-i.i.d. Environment

**MLCONFIGURATION** *#nodes: 10, architecture, see Section 9.2* **DATASET CONFIGURATION** *dataset: mnist, batch size: 5, training data distribution: 4 succeeding classes per node (every next node takes the 4 classes shifted by 1 class compared to the previous node) with 10 samples for each class, test samples per class: 10* **NEURAL NETWORK CONFIGURATION** *Optimizer: Adam, learning rate: 0.001, momentum: none, l2: 0.005* **TRAINING CONFIGURATION** ***#iterations**: under study up to 300, **GAR**: under study, **deep transfer**: under study, **communication pattern**: under study, asynchrony mode: none, #iterations before sending: 1, req. class overlap: 2* **ATTACK CONFIGURATION** *#attackers: 10, attack mode: all-label-flip*

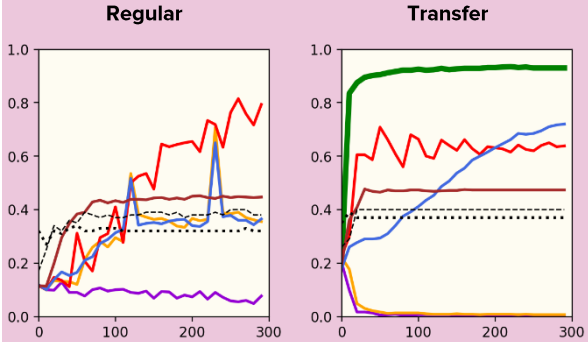



When the number of Byzantine attackers is low, we see that all GARs except for Pro-Bristle have a hard time to properly learn in the non-i.i.d. situations. However, when the number of attackers increases, Mozi manages to keep a stable performance limited to locally available classes of the peer, the performance of Krum gets extremely inconsistent, and only Pro-Bristle achieves a consistent high performance.

## Synchronous, Byzantine, non-i.i.d. Environment with varying comm. protocols

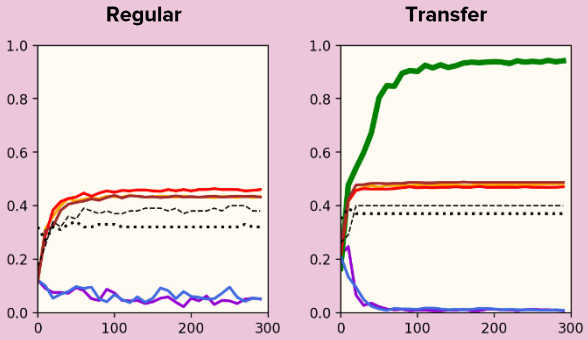
**MLCONFIGURATION** *#nodes: 10, architecture, see Section 9.2* **DATASET CONFIGURATION** *dataset: mnist, batch size: 5, training data distribution: 4 succeeding classes per node (every next node takes the 4 classes shifted by 1 class compared to the previous node) with 10 samples for each class, test samples per class: 10* **NEURAL NETWORK CONFIGURATION** *Optimizer: Adam, learning rate: 0.001, momentum: none, l2: 0.005* **TRAINING CONFIGURATION** ***#iterations****: under study up to 300,* ***GAR****: under study,* ***deep transfer****: under study,* ***communication pattern****: under study, asynchrony mode: none, #iterations before sending: 1, req. class overlap: 2* **ATTACK CONFIGURATION** *#attackers: 10, attack mode: all-label-flip*

### Communication to all nodes



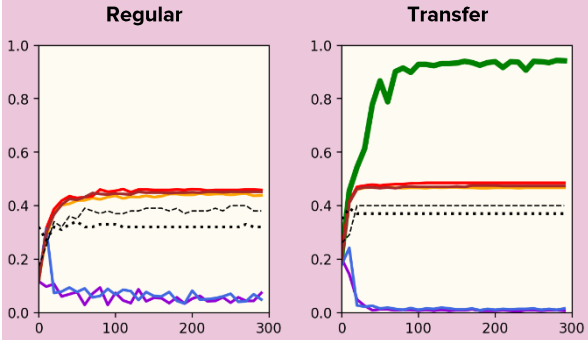

Total bandwidth regular GARs: 2159 MB
Total bandwidth Pro-Bristle: 648 MB

### Communication to random node



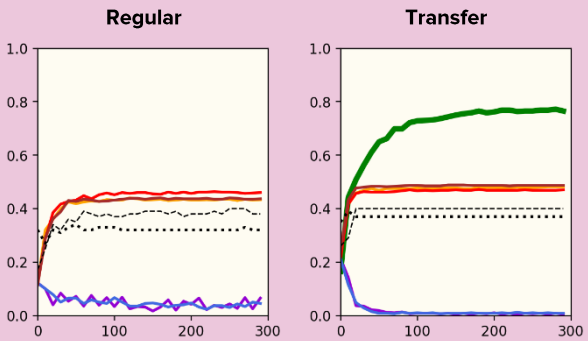

Total bandwidth regular GARs: 240 MB
Total bandwidth Pro-Bristle: 93 MB

### Round-robin communication




Total bandwidth regular GARs: 240 MB
Total bandwidth Pro-Bristle: 93 MB

### Ring communication



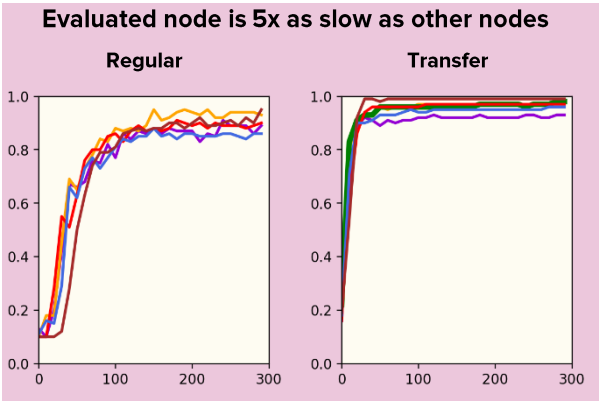

Total bandwidth regular GARs: 240 MB
Total bandwidth Pro-Bristle: 93 MB

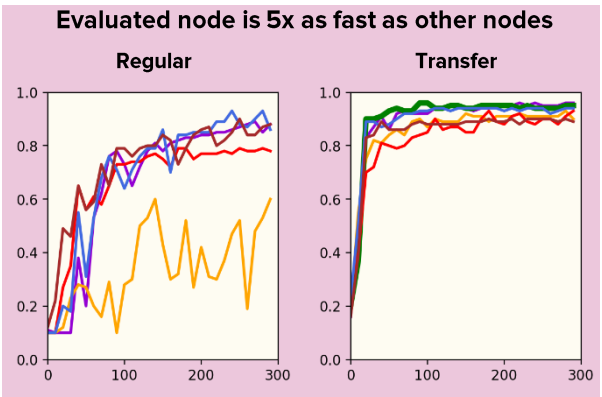
Suppose that we have a mildly Byzantine non-i.i.d. environment, does the network configuration matter? Clearly yes. As expected, the model converges fastest when all nodes communicate to all other nodes, but at the cost of considerable bandwidth usage. The three other communication patterns use the same (and much lower) amount of bandwidth, but also take considerably longer to converge. The reason why Krum and Bridge show similar performance as Mozi is that the nodes using them discard all models because they received too few models.

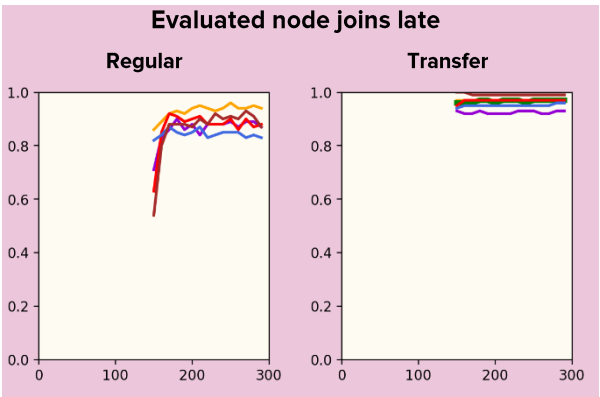
Average | Bridge | Median | Krum | Mozi | Pro-Bristle | Local min | Local max

## Synchronous, Byzantine Environment with various degrees of non-i.i.d.-ness

**MLCONFIGURATION** *#nodes: 10, architecture, see Section 9.2* **DATASET CONFIGURATION** *dataset: mnist, batch size: 5,* ***training data distribution****: under study, test samples per class: 10* **NEURAL NETWORK CONFIGURATION** *Optimizer: Adam, learning rate: 0.001, momentum: none, l2: 0.005* **TRAINING CONFIGURATION** ***#iterations****: under study up to 300,* ***GAR****: under study,* ***deep transfer****: under study,* ***communication pattern****: under study, asynchrony mode: none, #iterations before sending: 1, req. class overlap: 2* **ATTACK CONFIGURATION** *#attackers: 10, attack mode: all-label-flip*



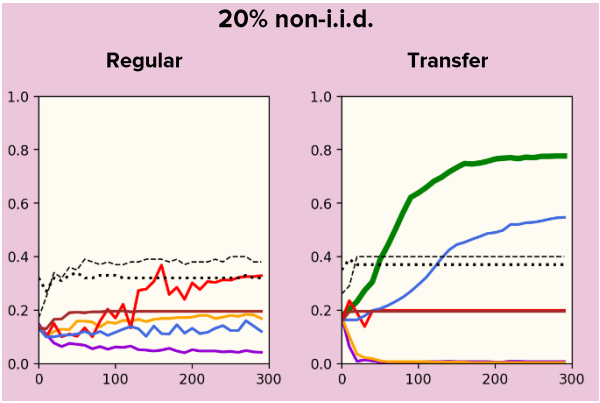



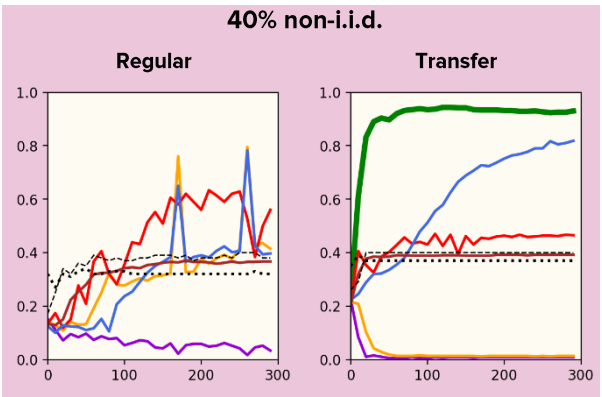



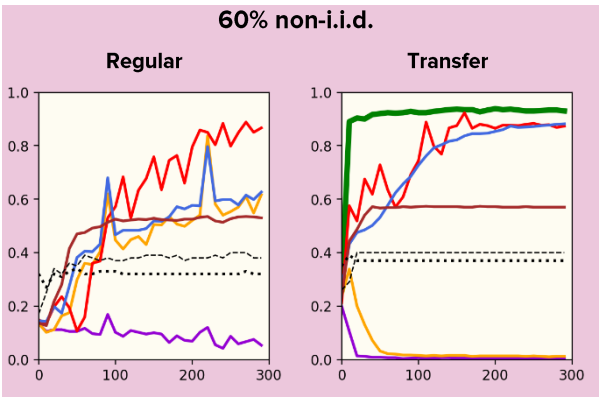

To check the asynchrony-resilience of the GARs, we evaluate the behavior of one particular node in 3 situations: (a) the evaluated node is 5x as slow in performing its iterations as the other nodes, (b) the evaluated node is 5x as fast as the other nodes, and (c) the evaluated node joins relatively late when all other nodes have already trained their model for 150 iterations. Although we expected that many GARs would not handle asynchrony well as data received from stale nodes could deteriorate the model, the GARs perform quite well in general. An exception is Bridge, which fails to ignore stale incoming models. This does not come as a surprise, since Bridge is highly dependent on a parameter that indicates the maximum number of Byzantine attacks (set to 2, see Section 9.8) whereas all models received are stale (and thus slightly Byzantine) when the node is faster than all other nodes. Pro-Bristle performs well in all three scenarios.

## Synchronous, Benign i.i.d. environment with various types of asynchrony

**MLCONFIGURATION** *#nodes: 10, architecture, see Section 9.2* **DATASET CONFIGURATION** *dataset: mnist, batch size: 5,* ***training data distribution****: under study, test samples per class: 10* **NEURAL NETWORK CONFIGURATION** *Optimizer: Adam, learning rate: 0.001, momentum: none, l2: 0.005* **TRAINING CONFIGURATION** ***#iterations****: under study up to 300,* ***GAR****: under study,* ***deep transfer****: under study,* ***communication pattern****: under study, asynchrony mode: none, #iterations before sending: 1, req. class overlap: 2* **ATTACK CONFIGURATION** *#attackers: 10, attack mode: all-label-flip*



## 10.1. Speed comparison

**Error! Reference source not found.** shows the time it took on our server to complete the experiments with 10 benign nodes, 4 Byzantine attackers using the all-label-flip attack, in a non-i.i.d. environment where each node has 40% of the classes and a Pro-Bristle requires a minimum class overlap of 3 classes. The running time of Average is obviously the fastest, followed closely by Median, Krum, and Bridge. Mozi takes somewhat longer because it has to evaluate the accuracy of each model. Pro-Bristle takes by far the longest, despite the fact that each node receives only 6 models instead of 15 models thanks to the PSI-CA filter. This longer runtime is caused by the performance integration that tests not only for each model but also for each class the performance.

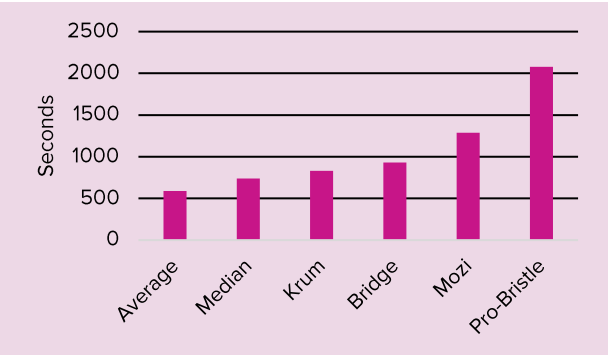



FIGURE 9 TIME TO COMPLETE THE ALL-LABEL-FLIP NON-I.I.D. EXPERIMENT

Figure 10 shows the number of seconds it took to run on our server the same experiment with 100 benign nodes instead of 10. Whereas Pro-Bristle was by far the slowest GAR is the former experiment, it is surprisingly fast in this experiment. This results from, on the one hand, the PSI-CA algorithm that eliminates 80% of the nodes that have insufficient class overlap to make the communication worthwhile, and on the other hand the fast distance-based filter that reduces the remaining candidates by 50% before those are passed to the expensive performance-based integration.

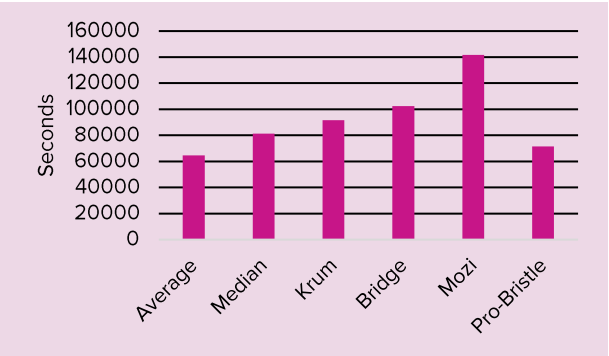



FIGURE 10 TIME TO COMPLETE THE ALL-LABEL-FLIP NON-I.I.D. EXPERIMENT

Average Bridge Median Krum Mozi Pro-Bristle Local min Local max al max

# 11. Discussion

Reputatie filter voor de distance-based filter => nog sneller

Meer variabelen evalueren

- #hidden layers
- Meer datasets
- Zonder momentum
- Met/zonder batch normalization

AR1 of andere non-i.i.d. technieken

Niet elke iteratie transmitten

Assumen vergelijkbaar dataset

Toepassing op non-classification problems

PSI-CA uitbreiden met ook meesturen hoeveel classes een peer in totaal heeft

- Voordeel dat als het er evenveel zijn als class overlap => geen andere parameters integreren
- Nadeel => andere node kan van weer een andere node goede foreign parameters hebben gekregen

Don't look at heterogeneity within a class (e.g. totally different cars are still cars)

## *11.1. 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 extremely hard 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 extremely 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 sent a message to another peer and then registered that it had send the message. However, when the CPU scheduler decided to pause the peer just after it had sent a message to another peer, then let the other peer respond, and then resume the execution of the former peer, then the response of the other peer was received before the next line (registering that the peer had sent the message to the other peer), causing the program to crash in a way that was terrible to debug.
- Another issue on which I could not find anything on the internet, and which took days to solve was that the debugger cannot attach to the emulator when Android Studio and IntelliJ Idea are running simultaneously. I have reported this bug to IntelliJ.
- Another issue that I came across that also seems to be limited to only my app 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 to proper deep-learning libraries in other languages. The best library currently available for Java is DeepLearning4j (DL4J), which is not being (seriously) maintained anymore[276]. Apart from the fact that fixing all dependencies took days (since all tutorials and documentation was 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-layer, I decided to divide the experiments over 16 separate emulators and run them on every emulator sequentially.

# 12. Conclusion

## 13. References


1. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V., 'How to Backdoor Federated Learning', in, *International Conference on Artificial Intelligence and Statistics*, (PMLR, 2020)
2. Bhagoji, A.N., Chakraborty, S., Mittal, P., and Calo, S., 'Analyzing Federated Learning through an Adversarial Lens', in, *International Conference on Machine Learning*, (PMLR, 2019)
3. Liu, Y., Ma, S., Aafer, Y., Lee, W.-C., Zhai, J., Wang, W., and Zhang, X., 'Trojaning Attack on Neural Networks', 2017.
4. Baruch, G., Baruch, M., and Goldberg, Y., 'A Little Is Enough: Circumventing Defenses for Distributed Learning', in, *Advances in neural information processing systems*, (2019)
5. Bhagoji, A.N., Chakraborty, S., Mittal, P., and Calo, S., '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)*, (2018)
6. Sun, Z., Kairouz, P., Suresh, A.T., and McMahan, H.B., 'Can You Really Backdoor Federated Learning?', *arXiv preprint arXiv:1911.07963*, 2019.
7. Xie, C., Huang, K., Chen, P.-Y., and Li, B., 'Dba: Distributed Backdoor Attacks against Federated Learning', in, *International Conference on Learning Representations*, (2019)
8. Chen, X., Liu, C., Li, B., Lu, K., and Song, D., 'Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning', *arXiv preprint arXiv:1712.05526*, 2017.
9. Koh, P.W. and Liang, P., 'Understanding Black-Box Predictions Via Influence Functions', *arXiv preprint arXiv:1703.04730*, 2017.
10. Suciu, O., Marginean, R., Kaya, Y., Daume III, H., and Dumitras, T., 'When Does Machine Learning {Fail}? Generalized Transferability for Evasion and Poisoning Attacks', in, *27th {USENIX} Security Symposium ({USENIX} Security 18)*, (2018)
11. Zou, M., Shi, Y., Wang, C., Li, F., Song, W., and Wang, Y., 'Potrojan: Powerful Neural-Level Trojan Designs in Deep Learning Models', *arXiv preprint arXiv:1802.03043*, 2018.
12. Gu, T., Dolan-Gavitt, B., and Garg, S., 'Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain', *arXiv preprint arXiv:1708.06733*, 2017.
13. Shafahi, A., Huang, W.R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., and Goldstein, T., 'Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks', in, *Advances in neural information processing systems*, (2018)
14. Shen, S., Tople, S., and Saxena, P., 'Auror: Defending against Poisoning Attacks in Collaborative Deep Learning Systems', in, *Proceedings of the 32nd Annual Conference on Computer Security Applications*, (2016)
15. Li, B., Wang, Y., Singh, A., and Vorobeychik, Y., 'Data Poisoning Attacks on Factorization-Based Collaborative Filtering', in, *Advances in neural information processing systems*, (2016)
16. Fung, C., Yoon, C.J., and Beschastnikh, I., 'Mitigating Sybils in Federated Learning Poisoning', *arXiv preprint arXiv:1808.04866*, 2018.
17. Tolpegin, V., Truex, S., Gursoy, M.E., and Liu, L., 'Data Poisoning Attacks against Federated Learning Systems', in, *European Symposium on Research in Computer Security*, (Springer, 2020)
18. Biggio, B., Nelson, B., and Laskov, P., 'Poisoning Attacks against Support Vector Machines', *arXiv preprint arXiv:1206.6389*, 2012.
19. Rubinstein, B.I., Nelson, B., Huang, L., Joseph, A.D., Lau, S.-h., Rao, S., Taft, N., and Tygar, J.D., 'Antidote: Understanding and Defending against Poisoning of Anomaly Detectors', in, *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, (2009)
20. Xiao, H., Biggio, B., Brown, G., Fumera, G., Eckert, C., and Roli, F., 'Is Feature Selection Secure against Training Data Poisoning?', in, *International Conference on Machine Learning*, (2015)
21. Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E.C., and Roli, F., 'Towards Poisoning of Deep Learning Algorithms with Back-Gradient Optimization', in, *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, (2017)
22. Lesort, T., Lomonaco, V., Stoian, A., Maltoni, D., Filliat, D., and Díaz-Rodríguez, N., 'Continual Learning for Robotics: Definition, Framework, Learning Strategies, Opportunities and Challenges', *Information fusion*, 2020, 58, pp. 52-68.
23. Li, S., Cheng, Y., Wang, W., Liu, Y., and Chen, T., 'Learning to Detect Malicious Clients for Robust Federated Learning', *arXiv preprint arXiv:2002.00211*, 2020.
24. Mohammad, U. and Sorour, S., 'Adaptive Task Allocation for Asynchronous Federated Mobile Edge Learning', *arXiv preprint arXiv:1905.01656*, 2019.
25. Xie, C., Koyejo, S., and Gupta, I., 'Asynchronous Federated Optimization', *arXiv preprint arXiv:1903.03934*, 2019.
26. Sprague, M.R., Jalalirad, A., Scavuzzo, M., Capota, C., Neun, M., Do, L., and Kopp, M., 'Asynchronous Federated Learning for Geospatial Applications', in, *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, (Springer, 2018)

27. Yang, Y.-R. and Li, W.-J., 'Basgd: Buffered Asynchronous Sgd for Byzantine Learning', *arXiv preprint arXiv:2003.00937*, 2020.
28. Chen, M., Mao, B., and Ma, T., 'Efficient and Robust Asynchronous Federated Learning with Stragglers', in, *Submitted to International Conference on Learning Representations*, (2019)
29. Hu, C., Jiang, J., and Wang, Z., 'Decentralized Federated Learning: A Segmented Gossip Approach', *arXiv preprint arXiv:1908.07782*, 2019.
30. Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., and Chandra, V., 'Federated Learning with Non-Iid Data', *arXiv preprint arXiv:1806.00582*, 2018.
31. Mhamdi, E.M.E., Guerraoui, R., and Rouault, S., 'The Hidden Vulnerability of Distributed Learning in Byzantium', *arXiv preprint arXiv:1802.07927*, 2018.
32. Wang, H., Sreenivasan, K., Rajput, S., Vishwakarma, H., Agarwal, S., Sohn, J.-y., Lee, K., and Papailiopoulos, D., 'Attack of the Tails: Yes, You Really Can Backdoor Federated Learning', *arXiv preprint arXiv:2007.05084*, 2020.
33. Xie, C., Koyejo, O., and Gupta, I., 'Generalized Byzantine-Tolerant Sgd', *arXiv preprint arXiv:1802.10116*, 2018.
34. Chen, L., Wang, H., Charles, Z., and Papailiopoulos, D., 'Draco: Byzantine-Resilient Distributed Training Via Redundant Gradients', *arXiv preprint arXiv:1803.09877*, 2018.
35. Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A., 'Signsgd: Compressed Optimisation for Non-Convex Problems', *arXiv preprint arXiv:1802.04434*, 2018.
36. Bernstein, J., Zhao, J., Azizzadenesheli, K., and Anandkumar, A., 'Signsgd with Majority Vote Is Communication Efficient and Fault Tolerant', *arXiv preprint arXiv:1810.05291*, 2018.
37. Sohn, J.-y., Han, D.-J., Choi, B., and Moon, J., 'Election Coding for Distributed Learning: Protecting Signsgd against Byzantine Attacks', *arXiv preprint arXiv:1910.06093*, 2019.
38. Gupta, N., Liu, S., and Vaidya, N.H., 'Byzantine Fault-Tolerant Distributed Machine Learning Using Stochastic Gradient Descent (Sgd) and Norm-Based Comparative Gradient Elimination (Cge)', *arXiv preprint arXiv:2008.04699*, 2020.
39. Tran, B., Li, J., and Madry, A., 'Spectral Signatures in Backdoor Attacks', in, *Advances in neural information processing systems*, (2018)
40. Zhao, L., Hu, S., Wang, Q., Jiang, J., Chao, S., Luo, X., and Hu, P., 'Shielding Collaborative Learning: Mitigating Poisoning Attacks through Client-Side Detection', *IEEE Transactions on Dependable and Secure Computing*, 2020.
41. Xie, C., Koyejo, S., and Gupta, I., 'Zeno: Distributed Stochastic Gradient Descent with Suspicion-Based Fault-Tolerance', in, *International Conference on Machine Learning*, (PMLR, 2019)
42. Xie, C., Koyejo, S., and Gupta, I., 'Zeno++: Robust Fully Asynchronous Sgd', *arXiv preprint arXiv:1903.07020*, 2019.
43. Rajput, S., Wang, H., Charles, Z., and Papailiopoulos, D., 'Detox: A Redundancy-Based Framework for Faster and More Robust Gradient Aggregation', in, *Advances in neural information processing systems*, (2019)
44. Jiang, Y., Wang, S., Ko, B.J., Lee, W.-H., and Tassiulas, L., 'Model Pruning Enables Efficient Federated Learning on Edge Devices', *arXiv preprint arXiv:1909.12326*, 2019.
45. Azulay, S., Raz, L., Globerson, A., Koren, T., and Afek, Y., 'Holdout Sgd: Byzantine Tolerant Federated Learning', *arXiv preprint arXiv:2008.04612*, 2020.
46. Mo, F. and Haddadi, H., 'Efficient and Private Federated Learning Using Tee', in, *EuroSys*, (2019)
47. Ji, J., Chen, X., Wang, Q., Yu, L., and Li, P., 'Learning to Learn Gradient Aggregation by Gradient Descent', in, *IJCAI*, (2019)
48. Shen, Y. and Sanghavi, S., 'Learning with Bad Training Data Via Iterative Trimmed Loss Minimization', in, *International Conference on Machine Learning*, (PMLR, 2019)
49. Wang, S., Tuor, T., Salonidis, T., Leung, K.K., Makaya, C., He, T., and Chan, K., 'Adaptive Federated Learning in Resource Constrained Edge Computing Systems', *IEEE Journal on Selected Areas in Communications*, 2019, 37, (6), pp. 1205-1221.
50. Kopparapu, K. and Lin, E., 'Fedfmc: Sequential Efficient Federated Learning on Non-Iid Data', *arXiv preprint arXiv:2006.10937*, 2020.
51. Chen, Z., Tian, P., Liao, W., and Yu, W., 'Zero Knowledge Clustering Based Adversarial Mitigation in Heterogeneous Federated Learning', *IEEE Transactions on Network Science and Engineering*, 2020.
52. Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V., 'Federated Optimization in Heterogeneous Networks', *arXiv preprint arXiv:1812.06127*, 2018.
53. Nilsson, A., Smith, S., Ulm, G., Gustavsson, E., and Jirstrand, M., 'A Performance Evaluation of Federated Learning Algorithms', in, *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, (2018)
54. Karimireddy, S.P., Kale, S., Mohri, M., Reddi, S.J., Stich, S.U., and Suresh, A.T., 'Scaffold: Stochastic Controlled Averaging for on-Device Federated Learning', *arXiv preprint arXiv:1910.06378*, 2019.
55. Yin, D., Chen, Y., Ramchandran, K., and Bartlett, P., 'Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates', *arXiv preprint arXiv:1803.01498*, 2018.

56. Fang, M., Cao, X., Jia, J., and Gong, N., 'Local Model Poisoning Attacks to Byzantine-Robust Federated Learning', in, *29th {USENIX} Security Symposium ({USENIX} Security 20)*, (2020)
57. Yang, Z., Gang, A., and Bajwa, W.U., 'Adversary-Resilient Inference and Machine Learning: From Distributed to Decentralized', *stat*, 2019, 1050, p. 23.
58. Chen, X., Chen, T., Sun, H., Wu, Z.S., and Hong, M., 'Distributed Training with Heterogeneous Data: Bridging Median- and Mean-Based Algorithms', *arXiv preprint arXiv:1906.01736*, 2019.
59. Cao, D., Chang, S., Lin, Z., Liu, G., and Sun, D., 'Understanding Distributed Poisoning Attack in Federated Learning', in, *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, (IEEE, 2019)
60. Yang, Z. and Bajwa, W.U., 'Byrdie: Byzantine-Resilient Distributed Coordinate Descent for Decentralized Learning', *IEEE Transactions on Signal and Information Processing over Networks*, 2019, 5, (4), pp. 611-627.
61. Yang, Z. and Bajwa, W.U., 'Bridge: Byzantine-Resilient Decentralized Gradient Descent', *arXiv preprint arXiv:1908.08098*, 2019.
62. Peng, J. and Ling, Q., 'Byzantine-Robust Decentralized Stochastic Optimization', in, *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (IEEE, 2020)
63. He, L., Karimireddy, S.P., and Jaggi, M., 'Byzantine-Robust Learning on Heterogeneous Datasets Via Resampling', *arXiv preprint arXiv:2006.09365*, 2020.
64. Jin, R., He, X., and Dai, H., 'Distributed Byzantine Tolerant Stochastic Gradient Descent in the Era of Big Data', in, *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, (IEEE, 2019)
65. Zhao, Y., Chen, J., Zhang, J., Wu, D., Teng, J., and Yu, S., 'Pdgan: A Novel Poisoning Defense Method in Federated Learning Using Generative Adversarial Network', in, *International Conference on Algorithms and Architectures for Parallel Processing*, (Springer, 2019)
66. Wang, Z., Song, M., Zhang, Z., Song, Y., Wang, Q., and Qi, H., 'Beyond Inferring Class Representatives: User-Level Privacy Leakage from Federated Learning', in, *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, (IEEE, 2019)
67. Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., and Zhao, B.Y., 'Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks', in, *2019 IEEE Symposium on Security and Privacy (SP)*, (IEEE, 2019)
68. Koh, P.W., Steinhardt, J., and Liang, P., 'Stronger Data Poisoning Attacks Break Data Sanitization Defenses', *arXiv preprint arXiv:1811.00741*, 2018.
69. Regatti, J. and Gupta, A., 'Befriending the Byzantines through Reputation Scores', *arXiv preprint arXiv:2006.13421*, 2020.
70. Shayan, M., Fung, C., Yoon, C.J., and Beschastnikh, I., 'Biscotti: A Ledger for Private and Secure Peer-to-Peer Machine Learning', *arXiv preprint arXiv:1811.09904*, 2018.
71. Chen, X., Ji, J., Luo, C., Liao, W., and Li, P., 'When Machine Learning Meets Blockchain: A Decentralized, Privacy-Preserving and Secure Design', in, *2018 IEEE International Conference on Big Data (Big Data)*, (IEEE, 2018)
72. Kim, Y.J. and Hong, C.S., 'Blockchain-Based Node-Aware Dynamic Weighting Methods for Improving Federated Learning Performance', in, *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, (IEEE, 2019)
73. Weng, J., Weng, J., Zhang, J., Li, M., Zhang, Y., and Luo, W., 'Deepchain: Auditable and Privacy-Preserving Deep Learning with Blockchain-Based Incentive', *IEEE Transactions on Dependable and Secure Computing*, 2019.
74. Zhao, Y., Zhao, J., Jiang, L., Tan, R., Niyato, D., Li, Z., Lyu, L., and Liu, Y., 'Privacy-Preserving Blockchain-Based Federated Learning for Iot Devices', *IEEE Internet of Things Journal*, 2020.
75. Chen, Y., Luo, F., Li, T., Xiang, T., Liu, Z., and Li, J., 'A Training-Integrity Privacy-Preserving Federated Learning Scheme with Trusted Execution Environment', *Information Sciences*, 2020, 522, pp. 69-79.
76. Steinhardt, J., Koh, P.W.W., and Liang, P.S., 'Certified Defenses for Data Poisoning Attacks', in, *Advances in neural information processing systems*, (2017)
77. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., and Srivastava, B., 'Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering', *arXiv preprint arXiv:1811.03728*, 2018.
78. Shoham, N., Avidor, T., Keren, A., Israel, N., Benditkis, D., Mor-Yosef, L., and Zeitak, I., 'Overcoming Forgetting in Federated Learning on Non-Iid Data', *arXiv preprint arXiv:1910.07796*, 2019.
79. Athey, S., The Impact of Machine Learning on Economics', *The Economics of Artificial Intelligence: An Agenda*, (University of Chicago Press, 2018)
80. Chen, H., Laine, K., and Rindal, P., 'Fast Private Set Intersection from Homomorphic Encryption', in, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, (2017)
81. Ventura, D. and Warnick, S., 'A Theoretical Foundation for Inductive Transfer', *Brigham Young University, College of Physical and Mathematical Sciences*, 2007, 19.
82. LeCun, Y., Bengio, Y., and Hinton, G., 'Deep Learning', *nature*, 2015, 521, (7553), pp. 436-444.

83. Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., and Rellermeyer, J.S., 'A Survey on Distributed Machine Learning', *ACM Computing Surveys (CSUR)*, 2020, 53, (2), pp. 1-33.
84. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., and Isard, M., 'Tensorflow: A System for Large-Scale Machine Learning', in, *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, (2016)
85. Medicare, C.f. and Medicaid Services, The Health Insurance Portability and Accountability Act of 1996 (Hipaa)', (1996)
86. Voigt, P. and Von dem Bussche, A., 'The Eu General Data Protection Regulation (Gdpr)', *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.
87. McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B.A., 'Communication-Efficient Learning of Deep Networks from Decentralized Data', in, *Artificial Intelligence and Statistics*, (PMLR, 2017)
88. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, accessed Date Accessed 2017 Accessed
89. Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D., 'Federated Learning for Mobile Keyboard Prediction', *arXiv preprint arXiv:1811.03604*, 2018.
90. Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., and Beaufays, F., 'Applied Federated Learning: Improving Google Keyboard Query Suggestions', *arXiv preprint arXiv:1812.02903*, 2018.
91. Chen, M., Mathews, R., Ouyang, T., and Beaufays, F., 'Federated Learning of out-of-Vocabulary Words', *arXiv preprint arXiv:1903.10635*, 2019.
92. Ramaswamy, S., Mathews, R., Rao, K., and Beaufays, F., 'Federated Learning for Emoji Prediction in a Mobile Keyboard', *arXiv preprint arXiv:1906.04329*, 2019.
93. Chen, M., Suresh, A.T., Mathews, R., Wong, A., Allauzen, C., Beaufays, F., and Riley, M., 'Federated Learning of N-Gram Language Models', *arXiv preprint arXiv:1910.03432*, 2019.
94. Yuan, B., Ge, S., and Xing, W., 'A Federated Learning Framework for Healthcare Iot Devices', *arXiv preprint arXiv:2005.05083*, 2020.
95. Leroy, D., Coucke, A., Lavril, T., Gisselbrecht, T., and Dureau, J., 'Federated Learning for Keyword Spotting', in, *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (IEEE, 2019)
96. Sim, K.C., Beaufays, F., Benard, A., Guliani, D., Kabel, A., Khare, N., Lucassen, T., Zadrazil, P., Zhang, H., and Johnson, L., 'Personalization of End-to-End Speech Recognition on Mobile Devices for Named Entities', in, *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, (IEEE, 2019)
97. Niknam, S., Dhillon, H.S., and Reed, J.H., 'Federated Learning for Wireless Communications: Motivation, Opportunities, and Challenges', *IEEE Communications Magazine*, 2020, 58, (6), pp. 46-51.
98. Chen, M., Poor, H.V., Saad, W., and Cui, S., 'Wireless Communications for Collaborative Federated Learning in the Internet of Things', *arXiv preprint arXiv:2006.02499*, 2020.
99. Lin, K.-Y. and Huang, W.-R., 'Using Federated Learning on Malware Classification', in, *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, (IEEE, 2020)
100. Sozinov, K., Vlassov, V., and Girdzijauskas, S., '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, 2018)
101. Nguyen, T.D., Marchal, S., Miettinen, M., Fereidooni, H., Asokan, N., and Sadeghi, A.-R., 'Dïot: A Federated Self-Learning Anomaly Detection System for Iot', in, *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, (IEEE, 2019)
102. Cetin, B., Lazar, A., Kim, J., Sim, A., and Wu, K., 'Federated Wireless Network Intrusion Detection', in, *2019 IEEE International Conference on Big Data (Big Data)*, (IEEE, 2019)
103. Lu, Y., Huang, X., Zhang, K., Maharjan, S., and Zhang, Y., 'Blockchain Empowered Asynchronous Federated Learning for Secure Data Sharing in Internet of Vehicles', *IEEE Transactions on Vehicular Technology*, 2020, 69, (4), pp. 4298-4311.
104. Samarakoon, S., Bennis, M., Saad, W., and Debbah, M., 'Federated Learning for Ultra-Reliable Low-Latency V2v Communications', in, *2018 IEEE Global Communications Conference (GLOBECOM)*, (IEEE, 2018)
105. Gulati, A., Aujla, G.S., Chaudhary, R., Kumar, N., and Obaidat, M.S., 'Deep Learning-Based Content Centric Data Dissemination Scheme for Internet of Vehicles', in, *2018 IEEE International Conference on Communications (ICC)*, (IEEE, 2018)
106. Lu, Y., Huang, X., Dai, Y., Maharjan, S., and Zhang, Y., 'Federated Learning for Data Privacy Preservation in Vehicular Cyber-Physical Systems', *IEEE Network*, 2020, 34, (3), pp. 50-56.
107. Liu, Y., James, J., Kang, J., Niyato, D., and Zhang, S., 'Privacy-Preserving Traffic Flow Prediction: A Federated Learning Approach', *IEEE Internet of Things Journal*, 2020.
108. Mowla, N.I., Tran, N.H., Doh, I., and Chae, K., 'Federated Learning-Based Cognitive Detection of Jamming Attack in Flying Ad-Hoc Network', *IEEE Access*, 2019, 8, pp. 4338-4350.

109. Liu, Y., Huang, A., Luo, Y., Huang, H., Liu, Y., Chen, Y., Feng, L., Chen, T., Yu, H., and Yang, Q., 'Fedvision: An Online Visual Object Detection Platform Powered by Federated Learning', in, *AAAI*, (2020)
110. Schneble, W. and Thamilarasu, G., 'Attack Detection Using Federated Learning in Medical Cyber-Physical Systems'.
111. Lu, S., Zhang, Y., and Wang, Y., 'Decentralized Federated Learning for Electronic Health Records', in, *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, (IEEE, 2020)
112. Brisimi, T.S., Chen, R., Mela, T., Olshevsky, A., Paschalidis, I.C., and Shi, W., 'Federated Learning of Predictive Models from Federated Electronic Health Records', *International journal of medical informatics*, 2018, 112, pp. 59-67.
113. Xu, J. and Wang, F., 'Federated Learning for Healthcare Informatics', *arXiv preprint arXiv:1911.06270*, 2019.
114. Rieke, N., Hancox, J., Li, W., Milletari, F., Roth, H., Albarqouni, S., Bakas, S., Galtier, M.N., Landman, B., and Maier-Hein, K., 'The Future of Digital Health with Federated Learning', *arXiv preprint arXiv:2003.08119*, 2020.
115. Konečný, J., McMahan, H.B., Ramage, D., and Richtárik, P., 'Federated Optimization: Distributed Machine Learning for on-Device Intelligence', *arXiv preprint arXiv:1610.02527*, 2016.
116. Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., and McMahan, H.B., 'Towards Federated Learning at Scale: System Design', *arXiv preprint arXiv:1902.01046*, 2019.
117. Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J., 'Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent', in, *Advances in neural information processing systems*, (2017)
118. Xie, X., Ma, L., Wang, H., Li, Y., Liu, Y., and Li, X., 'Diffchaser: Detecting Disagreements for Deep Neural Networks', in, *IJCAI*, (2019)
119. Dobbe, R., Fridovich-Keil, D., and Tomlin, C., 'Fully Decentralized Policies for Multi-Agent Systems: An Information Theoretic Approach', in, *Advances in neural information processing systems*, (2017)
120. Tang, H., Gan, S., Zhang, C., Zhang, T., and Liu, J., 'Communication Compression for Decentralized Training', in, *Advances in neural information processing systems*, (2018)
121. Lalitha, A., Wang, X., Kilinc, O., Lu, Y., Javidi, T., and Koushanfar, F., 'Decentralized Bayesian Learning over Graphs', *arXiv preprint arXiv:1905.10466*, 2019.
122. Nedic, A. and Ozdaglar, A., 'Distributed Subgradient Methods for Multi-Agent Optimization', *IEEE Transactions on Automatic Control*, 2009, 54, (1), pp. 48-61.
123. Hegedűs, I., Danner, G., and Jelasity, M., 'Decentralized Learning Works: An Empirical Comparison of Gossip Learning and Federated Learning', *Journal of Parallel and Distributed Computing*, 2021, 148, pp. 109-124.
124. Harinath, D., Satyanarayana, P., and Murthy, M., 'A Review on Security Issues and Attacks in Distributed Systems', *Journal of Advances in Information Technology*, 2017, 8, (1).
125. Lamport, L., Shostak, R., and Pease, M., The Byzantine Generals Problem', *Concurrency: The Works of Leslie Lamport*, (2019)
126. Blanchard, P., Guerraoui, R., and Stainer, J., 'Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent', in, *Advances in neural information processing systems*, (2017)
127. Chen, Y., Su, L., and Xu, J., 'Distributed Statistical Machine Learning in Adversarial Settings: Byzantine Gradient Descent', *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2017, 1, (2), pp. 1-25.
128. Zhang, Q., Cheng, L., and Boutaba, R., 'Algorithms and Architectures for Parallel Processing', *J. Int. Serv. Appl*, 2010, 1, (1), pp. 7-18.
129. Chen, J., Pan, X., Monga, R., Bengio, S., and Jozefowicz, R., 'Revisiting Distributed Synchronous Sgd', *arXiv preprint arXiv:1604.00981*, 2016.
130. Wu, W., He, L., Lin, W., Mao, R., Maple, C., and Jarvis, S.A., 'Safa: A Semi-Asynchronous Protocol for Fast Federated Learning with Low Overhead', *IEEE Transactions on Computers*, 2020.
131. Chen, Y., Ning, Y., and Rangwala, H., 'Asynchronous Online Federated Learning for Edge Devices', *arXiv preprint arXiv:1911.02134*, 2019.
132. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., and Cummings, R., 'Advances and Open Problems in Federated Learning', *arXiv preprint arXiv:1912.04977*, 2019.
133. Muñoz-González, L., Co, K.T., and Lupu, E.C., 'Byzantine-Robust Federated Machine Learning through Adaptive Model Averaging', *arXiv preprint arXiv:1909.05125*, 2019.
134. Dwork, C., 'Differential Privacy: A Survey of Results', in, *International conference on theory and applications of models of computation*, (Springer, 2008)
135. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., and Zhang, L., 'Deep Learning with Differential Privacy', in, *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, (2016)
136. Wei, K., Li, J., Ding, M., Ma, C., Yang, H.H., Farokhi, F., Jin, S., Quek, T.Q., and Poor, H.V., 'Federated Learning with Differential Privacy: Algorithms and Performance Analysis', *IEEE Transactions on Information Forensics and Security*, 2020, 15, pp. 3454-3469.

137. Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., and Li, B., 'Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning', in, *2018 IEEE Symposium on Security and Privacy (SP)*, (IEEE, 2018)
138. Biggio, B., Didaci, L., Fumera, G., and Roli, F., 'Poisoning Attacks to Compromise Face Templates', in, *2013 International Conference on Biometrics (ICB)*, (IEEE, 2013)
139. Yang, G., Gong, N.Z., and Cai, Y., 'Fake Co-Visitation Injection Attacks to Recommender Systems', in, *NDSS*, (2017)
140. Robbins, H. and Monro, S., 'A Stochastic Approximation Method', *The annals of mathematical statistics*, 1951, pp. 400-407.
141. Kingma, D.P. and Ba, J., 'Adam: A Method for Stochastic Optimization', *arXiv preprint arXiv:1412.6980*, 2014.
142. Mukkamala, M.C. and Hein, M., 'Variants of Rmsprop and Adagrad with Logarithmic Regret Bounds', *arXiv preprint arXiv:1706.05507*, 2017.
143. Damaskinos, G., El Mhamdi, E.M., Guerraoui, R., Guirguis, A.H.A., and Rouault, S.L.A., 'Aggregathor: Byzantine Machine Learning Via Robust Gradient Aggregation', in, *The Conference on Systems and Machine Learning (SysML), 2019*, (2019)
144. Zhang, S., Choromanska, A.E., and LeCun, Y., 'Deep Learning with Elastic Averaging Sgd', in, *Advances in neural information processing systems*, (2015)
145. Li, M., Andersen, D.G., Park, J.W., Smola, A.J., Ahmed, A., Josifovski, V., Long, J., Shekita, E.J., and Su, B.-Y., 'Scaling Distributed Machine Learning with the Parameter Server', in, *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, (2014)
146. Xing, E.P., Ho, Q., Xie, P., and Wei, D., 'Strategies and Principles of Distributed Machine Learning on Big Data', *Engineering*, 2016, 2, (2), pp. 179-195.
147. Choromanska, A., Henaff, M., Mathieu, M., Arous, G.B., and LeCun, Y., 'The Loss Surfaces of Multilayer Networks', in, *Artificial intelligence and statistics*, (PMLR, 2015)
148. Goodfellow, I.J., Vinyals, O., and Saxe, A.M., 'Qualitatively Characterizing Neural Network Optimization Problems', *arXiv preprint arXiv:1412.6544*, 2014.
149. Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y., 'Identifying and Attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization', *arXiv preprint arXiv:1406.2572*, 2014.
150. Kempe, D., Dobra, A., and Gehrke, J., 'Gossip-Based Computation of Aggregate Information', in, *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, (IEEE, 2003)
151. Hegedűs, I., Danner, G., and Jelasity, M., 'Decentralized Recommendation Based on Matrix Factorization: A Comparison of Gossip and Federated Learning', in, *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, (Springer, 2019)
152. Neglia, G., Xu, C., Towsley, D., and Calbi, G., 'Decentralized Gradient Methods: Does Topology Matter?', in, *International Conference on Artificial Intelligence and Statistics*, (PMLR, 2020)
153. Hegedűs, I., Danner, G., and Jelasity, M., 'Gossip Learning as a Decentralized Alternative to Federated Learning', in, *IFIP International Conference on Distributed Applications and Interoperable Systems*, (Springer, 2019)
154. El-Mhamdi, E.-M. and Guerraoui, R., 'Fast and Secure Distributed Learning in High Dimension', *arXiv preprint arXiv:1905.04374*, 2019.
155. Haykin, S., *Neural Networks and Learning Machines, 3/E*, (Pearson Education India, 2010)
156. Neelakantan, A., Vilnis, L., Le, Q.V., Sutskever, I., Kaiser, L., Kurach, K., and Martens, J., 'Adding Gradient Noise Improves Learning for Very Deep Networks', *arXiv preprint arXiv:1511.06807*, 2015.
157. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P.T.P., 'On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima', *arXiv preprint arXiv:1609.04836*, 2016.
158. Kleinberg, R., Li, Y., and Yuan, Y., 'An Alternative View: When Does Sgd Escape Local Minima?', *arXiv preprint arXiv:1802.06175*, 2018.
159. Bottou, L., 'Online Learning and Stochastic Approximations', *On-line learning in neural networks*, 1998, 17, (9), p. 142.
160. Guo, S., Zhang, T., Xie, X., Ma, L., Xiang, T., and Liu, Y., 'Towards Byzantine-Resilient Learning in Decentralized Systems', *arXiv preprint arXiv:2002.08569*, 2020.
161. Huber, P.J., *Robust Statistics*, (John Wiley & Sons, 2004)
162. Cretu, G.F., Stavrou, A., Locasto, M.E., Stolfo, S.J., and Keromytis, A.D., 'Casting out Demons: Sanitizing Training Data for Anomaly Sensors', in, *2008 IEEE Symposium on Security and Privacy (sp 2008)*, (IEEE, 2008)
163. Bhatia, K., Jain, P., and Kar, P., 'Robust Regression Via Hard Thresholding', in, *Advances in neural information processing systems*, (2015)
164. Diakonikolas, I., Kamath, G., Kane, D., Li, J., Moitra, A., and Stewart, A., 'Robust Estimators in High-Dimensions without the Computational Intractability', *SIAM Journal on Computing*, 2019, 48, (2), pp. 742-864.
165. Lai, K.A., Rao, A.B., and Vempala, S., 'Agnostic Estimation of Mean and Covariance', in, *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, (IEEE, 2016)

166. Su, L. and Vaidya, N.H., 'Fault-Tolerant Distributed Optimization (Part Iv): Constrained Optimization with Arbitrary Directed Networks', *arXiv preprint arXiv:1511.01821*, 2015.
167. Sundaram, S. and Gharesifard, B., 'Distributed Optimization under Adversarial Nodes', *IEEE Transactions on Automatic Control*, 2018, 64, (3), pp. 1063-1076.
168. Alon, N., Matias, Y., and Szegedy, M., 'The Space Complexity of Approximating the Frequency Moments', *Journal of Computer and system sciences*, 1999, 58, (1), pp. 137-147.
169. Jerrum, M.R., Valiant, L.G., and Vazirani, V.V., 'Random Generation of Combinatorial Structures from a Uniform Distribution', *Theoretical computer science*, 1986, 43, pp. 169-188.
170. Lerasle, M. and Oliveira, R.I., 'Robust Empirical Mean Estimators', *arXiv preprint arXiv:1112.3914*, 2011.
171. Minsker, S., 'Geometric Median and Robust Estimation in Banach Spaces', *Bernoulli*, 2015, 21, (4), pp. 2308-2335.
172. Minsker, S., 'Distributed Statistical Estimation and Rates of Convergence in Normal Approximation', *Electronic Journal of Statistics*, 2019, 13, (2), pp. 5213-5252.
173. Li, L., Xu, W., Chen, T., Giannakis, G.B., and Ling, Q., 'Rsa: Byzantine-Robust Stochastic Aggregation Methods for Distributed Learning from Heterogeneous Datasets', in, *Proceedings of the AAAI Conference on Artificial Intelligence*, (2019)
174. Barreno, M., Nelson, B., Joseph, A.D., and Tygar, J.D., 'The Security of Machine Learning', *Machine Learning*, 2010, 81, (2), pp. 121-148.
175. Cao, X. and Lai, L., 'Distributed Gradient Descent Algorithm Robust to an Arbitrary Number of Byzantine Attackers', *IEEE Transactions on Signal Processing*, 2019, 67, (22), pp. 5850-5864.
176. Mothukuri, V., Parizi, R.M., Pouriyeh, S., Huang, Y., Dehghantanha, A., and Srivastava, G., 'A Survey on Security and Privacy of Federated Learning', *Future Generation Computer Systems*, 2020.
177. Liu, K., Dolan-Gavitt, B., and Garg, S., 'Fine-Pruning: Defending against Backdooring Attacks on Deep Neural Networks', in, *International Symposium on Research in Attacks, Intrusions, and Defenses*, (Springer, 2018)
178. Qiao, M. and Valiant, G., 'Learning Discrete Distributions from Untrusted Batches', *arXiv preprint arXiv:1711.08113*, 2017.
179. Chou, E., Tramèr, F., Pellegrino, G., and Boneh, D., 'Sentinet: Detecting Physical Attacks against Deep Learning Systems', *arXiv preprint arXiv:1812.00292*, 2018.
180. Diakonikolas, I., Kamath, G., Kane, D., Li, J., Steinhardt, J., and Stewart, A., 'Sever: A Robust Meta-Algorithm for Stochastic Optimization', in, *International Conference on Machine Learning*, (2019)
181. Schmid, R., Pfitzner, B., Beilharz, J., Arnrich, B., and Polze, A., 'Tangle Ledger for Decentralized Learning', in, *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, (IEEE, 2020)
182. Kim, H., Kim, S.-H., Hwang, J.Y., and Seo, C., 'Efficient Privacy-Preserving Machine Learning for Blockchain Network', *IEEE Access*, 2019, 7, pp. 136481-136495.
183. Kim, H., Park, J., Bennis, M., and Kim, S.-L., 'Blockchained on-Device Federated Learning', *IEEE Communications Letters*, 2019, 24, (6), pp. 1279-1283.
184. Zhou, S., Huang, H., Chen, W., Zhou, P., Zheng, Z., and Guo, S., 'Pirate: A Blockchain-Based Secure Framework of Distributed Machine Learning in 5g Networks', *IEEE Network*, 2020.
185. Toyoda, K. and Zhang, A.N., 'Mechanism Design for an Incentive-Aware Blockchain-Enabled Federated Learning Platform', in, *2019 IEEE International Conference on Big Data (Big Data)*, (IEEE, 2019)
186. Majeed, U. and Hong, C.S., 'Flchain: Federated Learning Via Mec-Enabled Blockchain Network', in, *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, (IEEE, 2019)
187. Salah, K., Rehman, M.H.U., Nizamuddin, N., and Al-Fuqaha, A., 'Blockchain for Ai: Review and Open Research Challenges', *IEEE Access*, 2019, 7, pp. 10127-10149.
188. Bao, X., Su, C., Xiong, Y., Huang, W., and Hu, Y., 'Flchain: A Blockchain for Auditable Federated Learning with Trust and Incentive', in, *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*, (IEEE, 2019)
189. TOYODA, K., MATHIOPOULOS, P.T., and ZHANG, A.N., 'Novel Blockchain-Based Incentive-Aware Federated Learning Platform with Mechanism Design'.
190. Zhao, Y., Zhao, J., Jiang, L., Tan, R., and Niyato, D., 'Mobile Edge Computing, Blockchain and Reputation-Based Crowdsourcing Iot Federated Learning: A Secure, Decentralized and Privacy-Preserving System', *arXiv preprint arXiv:1906.10893*, 2019.
191. Kang, J., Xiong, Z., Niyato, D., Yu, H., Liang, Y.-C., and Kim, D.I., 'Incentive Design for Efficient Federated Learning in Mobile Networks: A Contract Theory Approach', in, *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*, (IEEE, 2019)
192. Kang, J., Xiong, Z., Niyato, D., Xie, S., and Zhang, J., 'Incentive Mechanism for Reliable Federated Learning: A Joint Optimization Approach to Combining Reputation and Contract Theory', *IEEE Internet of Things Journal*, 2019, 6, (6), pp. 10700-10714.
193. Preuveneers, D., Rimmer, V., Tsingenopoulos, I., Spooren, J., Joosen, W., and Ilie-Zudor, E., 'Chained Anomaly Detection Models for Federated Learning: An Intrusion Detection Case Study', *Applied Sciences*, 2018, 8, (12), p. 2663.

194. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N., 'Algorand: Scaling Byzantine Agreements for Cryptocurrencies', in, *Proceedings of the 26th Symposium on Operating Systems Principles*, (2017)
195. Zhan, Y., Li, P., Qu, Z., Zeng, D., and Guo, S., 'A Learning-Based Incentive Mechanism for Federated Learning', *IEEE Internet of Things Journal*, 2020.
196. Khan, L.U., Tran, N.H., Pandey, S.R., Saad, W., Han, Z., Nguyen, M.N., and Hong, C.S., 'Federated Learning for Edge Networks: Resource Optimization and Incentive Mechanism', *arXiv preprint arXiv:1911.05642*, 2019.
197. Yu, H., Liu, Z., Liu, Y., Chen, T., Cong, M., Weng, X., Niyato, D., and Yang, Q., 'A Fairness-Aware Incentive Scheme for Federated Learning', in, *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, (2020)
198. Hu, R. and Gong, Y., 'Trading Data for Learning: Incentive Mechanism for on-Device Federated Learning', *arXiv preprint arXiv:2009.05604*, 2020.
199. Zeng, R., Zhang, S., Wang, J., and Chu, X., 'Fmore: An Incentive Scheme of Multi-Dimensional Auction for Federated Learning in Mec', *arXiv preprint arXiv:2002.09699*, 2020.
200. Yu, H., Liu, Z., Liu, Y., Chen, T., Cong, M., Weng, X., Niyato, D., and Yang, Q., 'A Sustainable Incentive Scheme for Federated Learning', *IEEE Intelligent Systems*, 2020.
201. Le, T.H.T., Tran, N.H., Tun, Y.K., Nguyen, M.N., Pandey, S.R., Han, Z., and Hong, C.S., 'An Incentive Mechanism for Federated Learning in Wireless Cellular Network: An Auction Approach', *arXiv preprint arXiv:2009.10269*, 2020.
202. Cong, M., Yu, H., Weng, X., Qu, J., Liu, Y., and Yiu, S.M., 'A Vcg-Based Fair Incentive Mechanism for Federated Learning', *arXiv preprint arXiv:2008.06680*, 2020.
203. Ding, N., Fang, Z., and Huang, J., '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, 2020)
204. Lim, W.Y.B., Xiong, Z., Kang, J., Niyato, D., Zhang, Y., Leung, C., and Miao, C., '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*, (2020)
205. Lim, W.Y.B., Xiong, Z., Miao, C., Niyato, D., Yang, Q., Leung, C., and Poor, H.V., 'Hierarchical Incentive Mechanism Design for Federated Machine Learning in Mobile Networks', *IEEE Internet of Things Journal*, 2020.
206. Ng, K.L., Chen, Z., Zelei Liu, H.Y., Liu, Y., and Yang, Q., 'A Multi-Player Game for Studying Federated Learning Incentive Schemes'.
207. Pandey, S.R., Suhail, S., Tun, Y.K., Alsenwi, M., and Hong, C.S., 'An Incentive Design to Perform Federated Learning'.
208. Feng, S., Niyato, D., Wang, P., Kim, D.I., and Liang, Y.-C., 'Joint Service Pricing and Cooperative 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, 2019)
209. Sarikaya, Y. and Ercetin, O., 'Motivating Workers in Federated Learning: A Stackelberg Game Perspective', *IEEE Networking Letters*, 2019, 2, (1), pp. 23-27.
210. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., and Seth, K., 'Practical Secure Aggregation for Federated Learning on User-Held Data', *arXiv preprint arXiv:1611.04482*, 2016.
211. Sabt, M., Achemlal, M., and Bouabdallah, A., 'Trusted Execution Environment: What It Is, and What It Is Not', in, *2015 IEEE Trustcom/BigDataSE/ISPA*, (IEEE, 2015)
212. Data, D., Song, L., and Diggavi, S., 'Data Encoding for Byzantine-Resilient Distributed Optimization', *arXiv preprint arXiv:1907.02664*, 2019.
213. Roy, A.G., Siddiqui, S., Pölsterl, S., Navab, N., and Wachinger, C., 'Braintorrent: A Peer-to-Peer Environment for Decentralized Federated Learning', *arXiv preprint arXiv:1905.06731*, 2019.
214. Haseltalab, A. and Akar, M., 'Approximate Byzantine Consensus in Faulty Asynchronous Networks', in, *2015 American Control Conference (ACC)*, (IEEE, 2015)
215. Zhang, Y. and Yang, Q., 'A Survey on Multi-Task Learning', *arXiv preprint arXiv:1707.08114*, 2017.
216. Hertz, J.A., *Introduction to the Theory of Neural Computation*, (CRC Press, 2018)
217. Parisi, G.I., Tani, J., Weber, C., and Wermter, S., 'Lifelong Learning of Human Actions with Deep Neural Network Self-Organization', *Neural Networks*, 2017, 96, pp. 137-149.
218. Parisi, G.I., Tani, J., Weber, C., and Wermter, S., 'Lifelong Learning of Spatiotemporal Representations with Dual-Memory Recurrent Self-Organization', *Frontiers in neurorobotics*, 2018, 12, p. 78.
219. Rabinowitz, N.C., Desjardins, G., Rusu, A.-A., Kavukcuoglu, K., Hadsell, R.T., Pascanu, R., Kirkpatrick, J., and Soyer, H.J., Progressive Neural Networks', (Google Patents, 2017)
220. Lomonaco, V. and Maltoni, D., 'Core50: A New Dataset and Benchmark for Continuous Object Recognition', in, *Conference on Robot Learning*, (PMLR, 2017)
221. Maltoni, D. and Lomonaco, V., 'Continuous Learning in Single-Incremental-Task Scenarios', *Neural Networks*, 2019, 116, pp. 56-73.

222. Lomonaco, V., Maltoni, D., and Pellegrini, L., 'Rehearsal-Free Continual Learning over Small Non-Iid Batches', *arXiv preprint arXiv:1907.03799*, 2019.
223. Li, Z. and Hoiem, D., 'Learning without Forgetting', *IEEE transactions on pattern analysis and machine intelligence*, 2017, 40, (12), pp. 2935-2947.
224. Kemker, R., McClure, M., Abitino, A., Hayes, T., and Kanan, C., 'Measuring Catastrophic Forgetting in Neural Networks', in, *Proceedings of the AAAI Conference on Artificial Intelligence*, (2018)
225. Liu, X., Masana, M., Herranz, L., Van de Weijer, J., Lopez, A.M., and Bagdanov, A.D., 'Rotate Your Networks: Better Weight Consolidation and Less Catastrophic Forgetting', in, *2018 24th International Conference on Pattern Recognition (ICPR)*, (IEEE, 2018)
226. Ritter, H., Botev, A., and Barber, D., 'Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting', *arXiv preprint arXiv:1805.07810*, 2018.
227. Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W., and Zhang, B.-T., 'Overcoming Catastrophic Forgetting by Incremental Moment Matching', *arXiv preprint arXiv:1703.08475*, 2017.
228. Zenke, F., Poole, B., and Ganguli, S., 'Continual Learning through Synaptic Intelligence', in, *International Conference on Machine Learning*, (PMLR, 2017)
229. Robins, A., '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, 1993)
230. Robins, A., 'Catastrophic Forgetting, Rehearsal and Pseudorehearsal', *Connection Science*, 1995, 7, (2), pp. 123-146.
231. Gepperth, A. and Karaoguz, C., 'A Bio-Inspired Incremental Learning Architecture for Applied Perceptual Problems', *Cognitive Computation*, 2016, 8, (5), pp. 924-934.
232. Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C.H., 'Icarl: Incremental Classifier and Representation Learning', in, *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, (2017)
233. Ma, C., Konečný, J., Jaggi, M., Smith, V., Jordan, M.I., Richtárik, P., and Takáč, M., 'Distributed Optimization with Arbitrary Local Solvers', *Optimization Methods and Software*, 2017, 32, (4), pp. 813-848.
234. Jaggi, M., Smith, V., Takác, M., Terhorst, J., Krishnan, S., Hofmann, T., and Jordan, M.I., 'Communication-Efficient Distributed Dual Coordinate Ascent', *Advances in neural information processing systems*, 2014, 27, pp. 3068-3076.
235. Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A.S., 'Federated Multi-Task Learning', in, *Advances in neural information processing systems*, (2017)
236. Kumar, S., Dutta, S., Chatturvedi, S., and Bhatia, M., 'Strategies for Enhancing Training and Privacy in Blockchain Enabled Federated Learning', in, *2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM)*, (IEEE, 2020)
237. Yao, X. and Sun, L., 'Continual Local Training for Better Initialization of Federated Models', in, *2020 IEEE International Conference on Image Processing (ICIP)*, (IEEE, 2020)
238. Gonzalez, C., Sakas, G., and Mukhopadhyay, A., 'What Is Wrong with Continual Learning in Medical Image Segmentation?', *arXiv preprint arXiv:2010.11008*, 2020.
239. Ling, C.X. and Bohn, T., 'A Conceptual Framework for Lifelong Learning', *arXiv preprint arXiv:1911.09704*, 2019.
240. Lomonaco, V., 'Continual Learning with Deep Architectures', 2019.
241. Bellet, A., Guerraoui, R., Taziki, M., and Tommasi, M., 'Personalized and Private Peer-to-Peer Machine Learning', in, *International Conference on Artificial Intelligence and Statistics*, (PMLR, 2018)
242. Vanhaesebrouck, P., Bellet, A., and Tommasi, M., 'Decentralized Collaborative Learning of Personalized Models over Networks', in, *Artificial Intelligence and Statistics*, (PMLR, 2017)
243. Preston, A.R. and Eichenbaum, H., 'Interplay of Hippocampus and Prefrontal Cortex in Memory', *Current Biology*, 2013, 23, (17), pp. R764-R773.
244. 'Google Trends - Transfer Learning Popularity Worldwide'.
245. Lv, S., Ye, J., Yin, S., Cheng, X., Feng, C., Liu, X., Li, R., Li, Z., Liu, Z., and Zhou, L., 'Unbalanced Private Set Intersection Cardinality Protocol with Low Communication Cost', *Future Generation Computer Systems*, 2020, 102, pp. 1054-1061.
246. Pohlig, S. and Hellman, M., 'An Improved Algorithm for Computing Logarithms over Gf (P) and Its Cryptographic Significance (Corresp.)', *IEEE transactions on information theory*, 1978, 24, (1), pp. 106-110.
247. Shamir, A., Rivest, R.L., and Adleman, L.M., Mental Poker', *The Mathematical Gardner*, (Springer, 1981)
248. De Cristofaro, E., Gasti, P., and Tsudik, G., 'Fast and Private Computation of Cardinality of Set Intersection and Union', in, *International Conference on Cryptology and Network Security*, (Springer, 2012)
249. Holzapfel, K., Karl, M., Lotz, L., Carle, G., Djeffal, C., Fruck, C., Haack, C., Heckmann, D., Kindt, P.H., and Köppl, M., 'Digital Contact Tracing Service: An Improved Decentralized Design for Privacy and Effectiveness', *arXiv preprint arXiv:2006.16960*, 2020.
250. Sattler, F., Wiedemann, S., Müller, K.-R., and Samek, W., 'Robust and Communication-Efficient Federated Learning from Non-Iid Data', *IEEE transactions on neural networks and learning systems*, 2019, 31, (9), pp. 3400-3413.

251. Konečný, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., and Bacon, D., 'Federated Learning: Strategies for Improving Communication Efficiency', *arXiv preprint arXiv:1610.05492*, 2016.
252. Han, S., Mao, H., and Dally, W.J., 'Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding', *arXiv preprint arXiv:1510.00149*, 2015.
253. Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D., '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*, (2014)
254. De Sa, C., Feldman, M., Ré, C., and Olukotun, K., 'Understanding and Optimizing Asynchronous Low-Precision Stochastic Gradient Descent', in, *Proceedings of the 44th Annual International Symposium on Computer Architecture*, (2017)
255. Alistarh, D., Li, J., Tomioka, R., and Vojnovic, M., 'Qsgd: Randomized Quantization for Communication-Optimal Stochastic Gradient Descent', *arXiv preprint arXiv:1610.02132*, 2016, 1.
256. Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H., 'Terngrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning', *arXiv preprint arXiv:1705.07878*, 2017.
257. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y., 'Dorefa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients', *arXiv preprint arXiv:1606.06160*, 2016.
258. Konečný, J. and Richtárik, P., 'Randomized Distributed Mean Estimation: Accuracy Vs. Communication', *Frontiers in Applied Mathematics and Statistics*, 2018, 4, p. 62.
259. Strom, N., 'Scalable Distributed Dnn Training Using Commodity Gpu Cloud Computing', in, *Sixteenth Annual Conference of the International Speech Communication Association*, (2015)
260. Tsuzuku, Y., Imachi, H., and Akiba, T., 'Variance-Based Gradient Compression for Efficient Distributed Deep Learning', *arXiv preprint arXiv:1802.06058*, 2018.
261. Dryden, N., Moon, T., Jacobs, S.A., and Van Essen, B., 'Communication Quantization for Data-Parallel Training of Deep Neural Networks', in, *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, (IEEE, 2016)
262. Aji, A.F. and Heafield, K., 'Sparse Communication for Distributed Gradient Descent', *arXiv preprint arXiv:1704.05021*, 2017.
263. Chen, C.-Y., Choi, J., Brand, D., Agrawal, A., Zhang, W., and Gopalakrishnan, K., 'Adacomp: Adaptive Residual Gradient Compression for Data-Parallel Distributed Training', in, *Proceedings of the AAAI Conference on Artificial Intelligence*, (2018)
264. Tao, Z. and Li, Q., 'Esgd: Communication Efficient Distributed Deep Learning on the Edge', in, *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, (2018)
265. Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W.J., 'Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training', *arXiv preprint arXiv:1712.01887*, 2017.
266. Luping, W., Wei, W., and Bo, L., 'Cmfl: Mitigating Communication Overhead for Federated Learning', in, *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, (IEEE, 2019)
267. Hu, H., Wang, D., and Wu, C., 'Distributed Machine Learning through Heterogeneous Edge Systems', in, *Proceedings of the AAAI Conference on Artificial Intelligence*, (2020)
268. Wang, S., Tuor, T., Salonidis, T., Leung, K.K., Makaya, C., He, T., and Chan, K., 'When Edge Meets Learning: Adaptive Control for Resource-Constrained Distributed Machine Learning', in, *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, (IEEE, 2018)
269. Abbasi, M., Rajabi, A., Gagné, C., and Bobba, R.B., 'Towards Dependable Deep Convolutional Neural Networks (Cnns) with out-Distribution Learning', *arXiv preprint arXiv:1804.08794*, 2018.
270. https://www.cis.fordham.edu/wisdm/dataset.php, accessed Date Accessed
271. Saeed, A., Ozcelebi, T., and Lukkien, J., 'Multi-Task Self-Supervised Learning for Human Activity Detection', *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2019, 3, (2), pp. 1-30.
272. Stokkink, Q., Epema, D., and Pouwelse, J., 'A Truly Self-Sovereign Identity System', *arXiv preprint arXiv:2007.00415*, 2020.
273. Stokkink, Q. and Pouwelse, J., '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, 2018)
274. Pouwelse, J.A., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D.H., Reinders, M., Van Steen, M.R., and Sips, H.J., 'Tribler: A Social-Based Peer-to-Peer System', *Concurrency and computation: Practice and experience*, 2008, 20, (2), pp. 127-138.
275. Zeilemaker, N., Capotă, M., Bakker, A., and Pouwelse, J., 'Tribler: P2p Media Search and Sharing', in, *Proceedings of the 19th ACM international conference on Multimedia*, (2011)
276. 'What Happened to Deeplearning4j?'.