# The Landscape of Modern Machine Learning: A Review of Machine, Distributed and Federated Learning

Omer Subasi[1*], Oceane Bel[1], Joseph Manzano[1], Kevin Barker[1]

[1*]High Performance Computing Group, Pacific Northwest National Laboratory, 902 Battelle Blvd, Richland, 99354, WA, USA.

*Corresponding author(s). E-mail(s): omer.subasi@pnnl.gov;
Contributing authors: obel@pnnl.gov; joseph.manzano@pnnl.gov; kevin.barker@pnnl.gov;

**Abstract**

With the advance of the powerful heterogeneous, parallel and distributed computing systems and ever increasing immense amount of data, machine learning has become an indispensable part of cutting-edge technology, scientific research and consumer products. In this study, we present a review of modern machine and deep learning. We provide a high-level overview for the latest advanced machine learning algorithms, applications, and frameworks. Our discussion encompasses parallel distributed learning, deep learning as well as federated learning. As a result, our work serves as an introductory text to the vast field of modern machine learning.

**Keywords:** Machine Learning, Distributed Machine Learning, Deep Learning, Federated Learning, Parallel and Distributed Computing.

## 1 Introduction

Over the last decade, Machine Learning (ML) has been applied to ever increasing immense amount of data that is becoming available as more people become daily users of internet, mobile and wireless networks. Coupled with the significant advances in deep learning (DL), ML has found more complex applications: from medical to machine translation and speech recognition, to intelligent object recognition, and to smart cities [1, 2]. Modern parallel and heterogeneous computing systems [3, 4, 5] have enabled such applications by supporting highly parallel training. These large-scale and distributed systems therefore have become the backbone of modern ML [6, 7, 8].

Federated Learning (FL), as a sub-field of DL, has emerged as a distributed learning solution to provide data privacy [9]. Ever since its inception [9], FL has been studied extensively and adapted widely [10, 11, 12, 13].

In this study, we review the current landscape of modern ML systems and applications, and offer an overview as a self-contained text. While there are many surveys on large-scale [6, 8], distributed ML [7], DL [1, 2, 14], and FL [10, 11, 12, 13], we instead provide a high-level joint view of modern parallel and distributed ML and FL. In this way, our work differentiates itself from the existing literature. In brief, our study

- presents the concepts and methods of ML and DL.
- discusses the parallelism and scaling approaches of large-scale distributed ML. Moreover, it explores the communication aspects, such as

costs, topologies, and networking, of parallel and distributed training and inference.
- introduces FL, its applications and aggregation methods. It then elaborates on the security and privacy aspects as well as the existing platforms and datasets.
- summarizes open research questions in the modern landscape of parallel and distributed ML, DL and FL.

Figure 1 outlines and summarizes our study.

Our study is organized as follows: Section 2 overviews the related work on large-scale and distributed ML. Section 3 provides the background on ML. Section 4 discusses distributed ML. Section 5 presents FL. Section 6 summarizes the existing open challenges. Finally, Section 7 concludes our review.

# 2 Related Work

Surveys pertaining to parallel, distributed and large scale ML have been very numerous in the literature [6, 7, 8]. Our work is different and unique because it provides an introductory review of the latest joint landscape of ML, DL and FL.

Different than the general surveys such as [15, 6, 7, 8], some surveys offer in depth cost and comparisons of algorithms and methods both theoretically and empirically [16, 17].

Many studies focus on distributed DL. Some of them are [1, 2, 18, 19, 14, 20]. Moreover, there exists a significant number of surveys that focus on specific types of models such as [21] for graph neural networks (GNNs), [22] for Internet-of-Things (IoTs), [23] for wireless networks, [24] for mobile and 5G networks or for specific target environments such as [25] for unmanned aerial vehicles (UAV).

FL literature unsurprisingly offers many surveys. Some of the latest surveys are [10, 11, 12]. Among studies having specific topics, [26] surveys privacy and security methods for FL, [27] discusses block chain-based FL. [28] presents differential privacy for FL. [13] offers a survey of FL for IoT.

# 3 Machine Learning (ML)

In this section, we first overview ML in terms of concepts and goals. Then we review various ML algorithms. Finally, we discuss the existing modern ML frameworks.

## 3.1 Introduction to ML

ML is the process of learning from data to perform complex tasks for which there is no known deterministic and algorithmic solution, or building such a solution is not practical. For instance, developing a deterministic algorithm based on rules to detect spam emails is highly impractical. It is not possible to know the exact list of the detection rules. In addition, these rules most often change over time. Since the list of the rules may be ever-increasing and even contradictory, the maintenance of such algorithms would require constant labor.

The ML process is mainly two-fold: Training and prediction (inference). In the training phase, the parameters of a learning model are optimized based on data. In the prediction phase, the trained model is deployed to perform predictions on new data. While in most cases the training and prediction phases are mutually exclusive, in incremental learning cases, they are coupled together. The models in these cases are continuously trained and make predictions. Figure 2 visualizes the training and prediction phases.

The main goal of ML is to generalize such that it performs well with unseen data. However, this goal contradicts its optimization goal in which ML tries to minimize the training loss with the training data. As a result, the well-known bias-variance problem emerges. If an ML model over-fits the training data, that is, having high variance, it performs poorly with the unseen data. On the other hand, if the model under-fits, that is, having high bias, it does not learn important patterns or regularities in the data. Over-fitting typically happens when a model is too complex for the underlying problem. In contrast, under-fitting happens when the model is too simple. Figure 3 depicts the bias-variance trade-off.

In the following, we present different types of ML tasks. After that, we look into different problems that ML can solve. Then, we review widely used ML algorithms and methods. Finally, we survey the existing ML platforms that are not supported with specialized hardware and not suited for DL or FL.

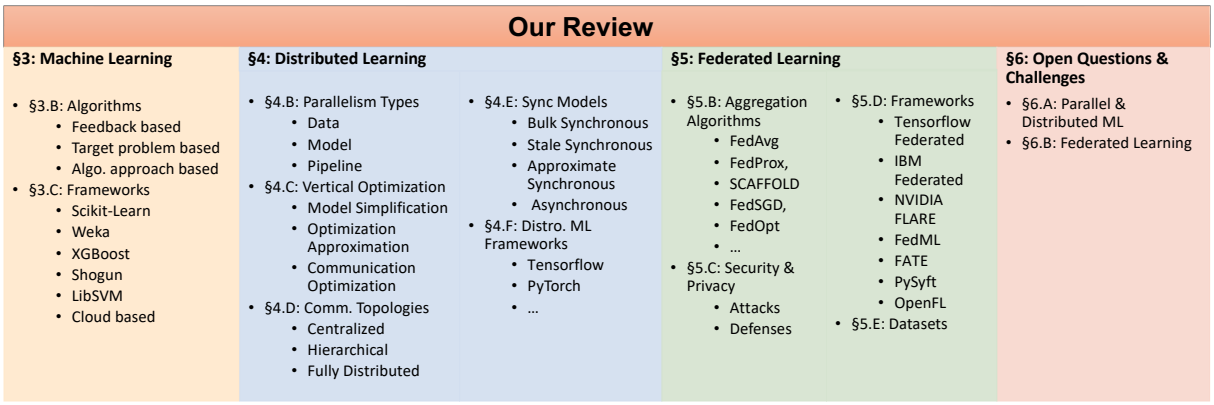



**Fig. 1**: The outline of our review.

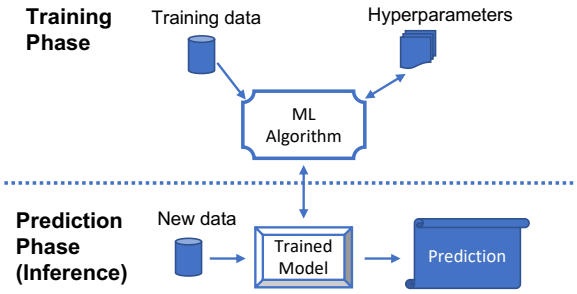



**Fig. 2**: ML phases: Training and prediction (inference).

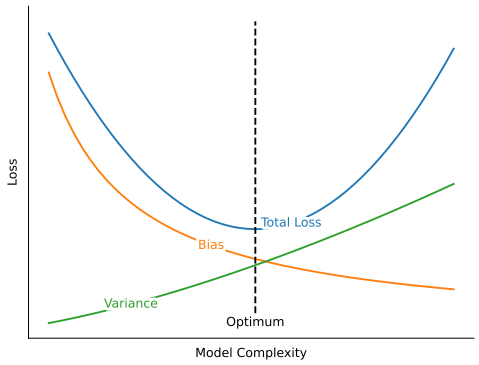



**Fig. 3**: Bias-variance trade-off. Model complexity with respect to bias and variance.

## 3.2 ML Algorithms

ML algorithms can be categorized by the format and requirements of data (external feedback), by the type of problems they are designed for (target problem), and by the techniques they use (algorithmic approaches).

It is worth noting that there is another way of categorizing ML: online and offline. In offline learning, the entire training data is available prior to training. This is the most common application of ML. In online learning [29], either the entire data is not available beforehand or it is computationally infeasible to perform training over the entire data at once. An example of the former is sequential training such as time series analysis in financial markets. An example of the latter is learning with a very large dataset which does not fit into the memory and consequently, training becomes prohibitive.

### 3.2.1 External feedback

ML algorithms can be classified based on the external feedback as follows:

Supervised Learning: Learning is performed by feeding labeled input data so that a model's parameters are optimized. Labeled data can be desired classes, categories, or numerical outputs corresponding to the training instances. During training, the optimization is achieved by minimizing a predetermined cost function. After training, the trained model is deployed to predict the outputs of new instances. An example supervised learning is to classify newly seen handwritten digits by training with the labeled digits.

Unsupervised Learning: The goal of unsupervised learning is to find structures and patterns

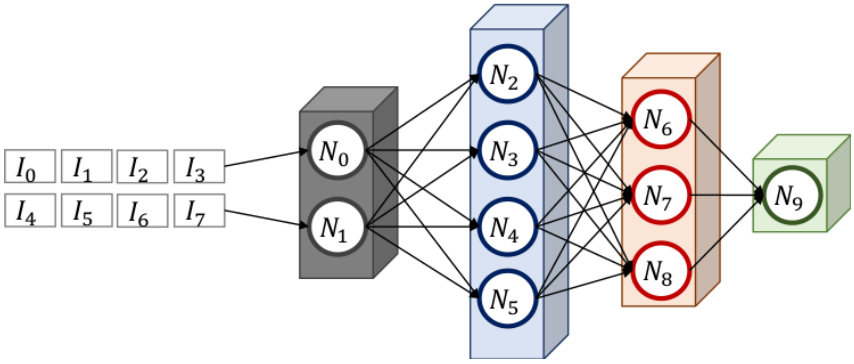



**Fig. 4**: An artificial neural network example.

in unlabeled data. This means that in unsupervised learning, the data does not possess desired outputs. As an example of unsupervised learning, clustering aims to find similar groups (clusters) in given data. Dimensionality reduction is another example of unsupervised learning where the goal is to find a subset of key features that describes the data well.

Semi-supervised Learning: In semi-supervised learning, the amount of labeled data is small while the amount of unlabeled data is large. Clustering algorithms are typically used to propagate existing labels to the unlabeled data. An assumption of semi-supervised learning is that similar data shares the same label.

Reinforcement Learning: Reinforcement learning is applied when an agent interacts with an environment. Based on the observations it makes, the agent takes actions. The actions are rewarded or penalized according to a reward function. Applications of reinforcement learning lie in the fields such as game theory, robotics and industrial automation.

### 3.2.2 Target Problem

Under this categorization, ML algorithms are grouped according to the kind of problems they are designed to solve.

In classification problems, the aim is to correctly categorize data instances into the known classes.

In regression problems, the goal is to estimate the value of a variable based on other variables (features).

Clustering finds the distinct groups of similar data instances based on a selected similarity metric.

Anomaly and novelty detection is used to find data instances that are significantly different than others. These instances are called outliers. In anomaly detection, training data consists of both outliers and regular (expected) data instances. In novelty detection, on the other hand, the goal is used to detect unseen data where training data is free of outliers.

Dimensionality reduction is used to reduce the number features of the training data. In dimensionality reduction, if a subset of the original set of the features is selected, it is called feature selection. In contrast, if features are combined into new ones, it is called feature extraction. Dimensionality reduction can also be used to decrease computational costs of training. Furthermore, it can also be used to prevent over-fitting. The problem of over-fitting with high-dimensional data is famously known as the curse of dimensionality. The curse of dimensionality arises due to data sparsity in high dimensional spaces.

### 3.2.3 Algorithmic Approaches

ML algorithms can be categorized based on algorithmic approaches that they employ.

Stochastic Gradient Decent (SGD) based algorithms are optimized based on a loss function of the outputs of the model parameters in the opposite direction of the gradient. Because at each training step a random subset of data is used, this optimization method is called stochastic. Many common ML algorithms are optimized with SGD such as artificial neural networks.

Support Vector Machines (SVMs) [30] are typically used when the input data is not linearly separable in its original space. They map the input data to high dimensional spaces where it becomes linearly separable. SVMs can be used for classification, regression, and novelty detection.

Artificial Neural Networks (ANNs) are constructed by multiple layers of nodes (neurons) that have inputs, outputs, corresponding feature weights, and an activation function. Layers can be input, hidden, and output layers. ANNs have recently been very successful in tasks such as image classification, object detection, and natural language processing. Figure 4 depicts an example of an ANN. Some well-known types of ANNs include:

- Convolutional Neural Networks (CNNs) [31] are deep neural networks that incorporate convolutions and pooling. While convolutions help with learning local data, pooling help with learning abstract features. CNNs have been extremely successful in tasks such as image classification, object detection, and image segmentation.
- Recurrent Neural Networks (RNNs) [32] maintains a temporal state of sequence data. The temporal state may hold short-term or long-term memory. RNNs are used in tasks such as time series forecasting, natural language processing, and anomaly detection.
- Autoencoders [33] are ANNs that learn latent representations of input data with no supervision. They are used for dimensionality reduction and visualization of high dimensional data.
- Generative Adversarial Networks (GANs) [34] are (originally unsupervised) neural networks used to generate data based on a game between a generator and discriminator network. They have been successfully applied in supervised and semi-supervised learning.
- Graph Neural Networks (GNNs) [35] are a type of ANNs designed to perform learning and prediction on data described by graphs. GNNs provide an easy way to do node, edge, and graph level ML tasks.
- Self-Organizing Maps (SOMs) [36] are neural networks which produce a low dimensional representation of high dimensional data. SOMs are used for visualization, clustering, and classification. The training is unsupervised where after random initialization, neurons compete against each other.
- Boltzmann Machines [37, 38] are fully connected ANNs which, unlike other ANNs, have probabilistic activation functions. Neurons output 1 or 0 based on Boltzmann distribution. Boltzmann Machines can be used for classifying, denoising, or completing images.
- Deep Belief Networks [39] are stacked Boltzmann Machines designed to tackle larger and more complex learning challenges. They are used for semi-supervised learning.
- Hopfield Networks [40, 41] are fully connected networks that are used for tasks such as character recognition.

Transformers [42] are a class of DL models that has shown extraordinary success in many ML fields including natural language processing and computer vision. Transformers were first introduced by a landmark paper from Google [43] which were based on a novel mechanism called *Attention*. At its core, a transformer is an encoder-decoder model. The success of Transformers has become a regular news-headliner such as the release of GPT-4 [44] and ChatGPT [45].

Rule-based algorithms [46] use a set of rules to learn patterns from the input data. They are typically easier to interpret than other ML algorithms. Decision trees are the most well-known rule-based algorithms.

Evolutionary algorithms [47] use ideas from biological evolution. In evolutionary algorithms, the target problem is represented by a set of properties. The performance metric is called fitness function. Based on fitness scores, the set of properties is mutated and crossed over. These algorithms iterate until accurate estimates are obtained. Evolutionary algorithms can also be used to create other algorithms such as neural networks.

Semantic and Topic algorithms [48, 49] are used to learn specific semantic patterns and distinct relationships in the input data. An example application of these algorithms is to find the topics and relate them to each other in a given set of documents.

Ensemble algorithms combine other algorithms to obtain a solution that performs better than the individual algorithms. Different ways to build ensembles are:

- Bagging combines multiple classifiers and uses voting to determine the final output.
- Boosting is a technique that trains the subsequent models with the data instances misclassified by the preceding models in the chain.
- Stacking is the process where a model trains with the outputs of the preceding models in a chain of several models. Stacking typically reduces the classification variance.
- Random Forests combine multiple decision trees and output an (weighted) average of the outputs of the individual trees.

### 3.3 Existing ML Frameworks

In this section, we present the existing ML platforms that are not supported with specialized hardware and typically not suited for DL or FL. We then briefly mention the popular ML services in the cloud.

Scikit-Learn [50] is the most popular open-source Python library that offers an extensive suite of ML algorithms. The library is very well maintained and provides a comprehensive set of algorithms, methods, pre-processing, pipelining, model selection and hyper-parameter search capabilities. It provides interfaces to work with NumPy and SciPy packages.

Weka [51] is a general-purpose and popular Java ML library. It provides a large collection of algorithms and visualization tools. Weka supports numerous tasks such as pre-processing, classification, regression, clustering and visualization.

XGBoost [52] is a scalable and distributed gradient boosting library based on decision trees. It implements parallel ML algorithms for classification, regression and ranking tasks.

Shogun [53] is a research-oriented open-source ML library. It offers a large number of ML algorithms and cross-platform support by providing bindings with other languages and environments such as Python, Octave, R, Java. Shogun's core library is implemented in C++.

LibSVM [54] is a specialized C/C++ library for SVMs. It provides interfaces for Python, R, MATLAB and many others.

Many companies offer standard ML and distributed ML services. Moreover, these services often include the support for GPUs and other ML specific hardware. Popular cloud ML services are Google's Cloud [55], Microsoft Azure [56], Amazon's SageMaker [57] and the IBM Watson Cloud [58].

## 4 Distributed Machine Learning

In this section, we introduce large-scale distributed ML. We then explore different types of parallelisms used in distributed training. Next, we dive into vertical scaling techniques. After that, we present the optimizations for communications in distributed ML. Then we continue with the communication topologies and synchronization models. Finally, we conclude this section by the discussion of the existing distributed ML frameworks.

As a side note, we use *client* and *participant* interchangeably in the rest of the paper.

### 4.1 Introduction to Distributed ML

Distributed ML is proposed to utilize distributed and heterogeneous computing systems to solve large and complex problems where a solution cannot be obtained by a single standalone homogeneous computing device. Distributed ML offers two different approaches. The first is to use heterogeneous resources available in a single computing system such as Graphical Processing Units (GPUs). This is called vertical scaling. The second is to use multiple machines to solve larger problems and to support fault-tolerance. This is called horizontal scaling.

GPUs have been the most common mean of vertical scaling. Given sufficient parallelism, it has been shown that GPUs significantly accelerate training [59, 60]. For instance, NVIDIA GPUs have been popular in accelerating ML [59, 61]. Vendors such as Google have implemented their own specific hardware accelerators. Tensor Processing Units (TPUs) [60] are designed specifically for this purpose. Others such as Graphcore [62] and SambaNova [63] have followed this trend with sophisticated dataflow-based hardware designs and powerful system software tool-chains.

In contrast to vertical scaling, horizontal scaling corresponds to distributed training and inference across multiple machines. Horizontal scaling enables ML solutions to handle applications and

data that do not fit in the resources of a single machine. Additionally, the usage of multiple machines typically accelerate training and inference.

## 4.2 Parallelisms in Distributed Training and Inference

There are three types of parallelisms used in distributed training. These are data, model and pipeline parallelism.

### 4.2.1 Data Parallelism

In data parallelism, the same ML model is trained with different subsets of the data in parallel at different computing resources. Once all computing resources finish the assigned training, the models are accumulated and an average model is obtained. Then, this average model is distributed back to each computing resource for the subsequent rounds of training. Figure 5 depicts data parallelism with two parallel resources.

The main advantage of data parallelism is that it is applicable to any distributed ML model without requiring expert/domain knowledge. It is also very scalable for compute-intensive models, such as CNNs. One disadvantage of data parallelism is that model synchronization may become a bottleneck. Another disadvantage occurs when the model does not fit in the memory of a single device.

### 4.2.2 Model Parallelism

In model parallelism, the model is partitioned and distributed to different computing resources. The data is distributed as well according to the model distribution. When there is a dependency among the computing resources, synchronization is needed for the parameters (weights) to be shared consistently. Figure 6 shows model parallelism where two resources are used. An important note is that in the figure, every time that a dashed line crosses a resource boundary, at least one synchronization event must take place to ensure data consistency.

The main advantage of model parallelism is that models take less memory in each single resource (device). Its main disadvantage is that the model partitioning is often nontrivial. Another disadvantage is the potential intensive communications among the resources.

### 4.2.3 Pipeline Parallelism

Pipeline parallelism combines model and data parallelisms. It distributes the model and data in a such a way that there is a pipeline among the computing resources in which each resource has a different part of the model. Pipeline parallelism maintains the advantages of model parallelism while increasing the resource utilization. Figure 7 illustrates pipeline parallelism.

## 4.3 Vertical Optimization Approaches

We have discussed the types of parallelisms used in distributed ML above. Now, we explore three vertical optimization approaches. They are model simplification, optimization approximation, and communication optimization approaches.

### 4.3.1 Model Simplification

Model simplification refers to the reformulation of a target model to decrease its computational complexity as a way of achieving efficiency. Model simplification can be further divided into categories based on the type of the ML models. These models can be based on kernels, trees, graphs and deep neural networks. Table 1 summarizes the model simplification techniques.

Simplifications for kernel-based models are made by sampling-based or projection-based approximations. While sampling-based methods [64, 65] approximate kernel matrices by random samples, projection-based methods [66, 67] use Gaussian or sparse random projections to map the data features to low dimensional sub-spaces.

Performance and scalability improvements for tree-based models, such as decision trees and random forests, are commonly based on rule [68] or feature sampling [52] [69].

Graph-based simplifications are developed for graph-based models where nodes represent the data instances and edges represent the similarity between the instances. In these models, the cost of training comes from two main sources: graph construction and the label matrix inversion. For sparse graphs, graph construction constitutes the main cost of training. This is because when label

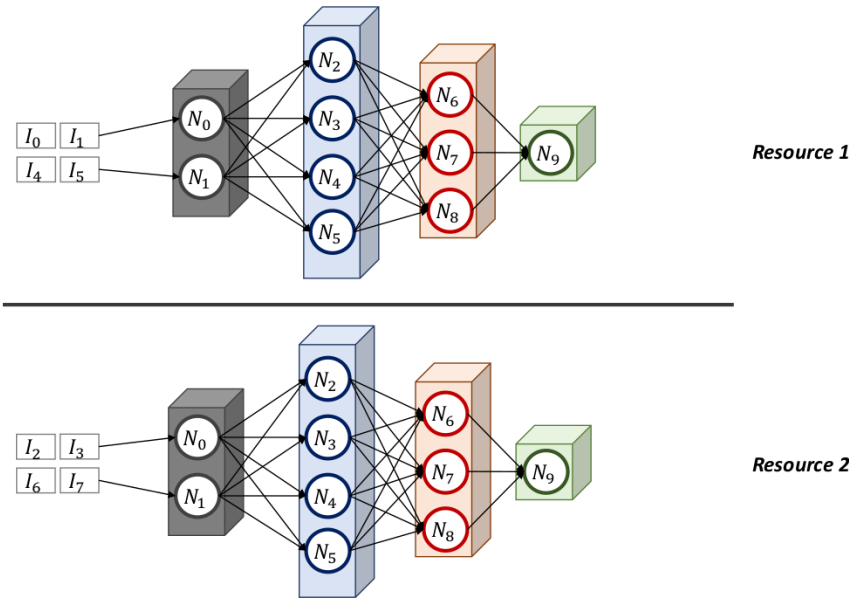



**Fig. 5**: Data parallelism for a deep neural network.

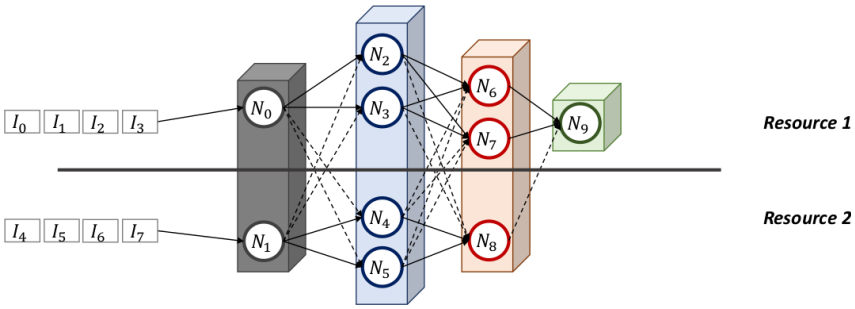



**Fig. 6**: Model parallelism for a deep neural network.

propagation is used, it lowers the cost of the inversion of the label matrix and it becomes less costly than graph construction. As a result, graph construction dominates the main computational cost. To construct sparse graphs [70], hashing methods [71] [72] are often used. Different than sparse graph models, there are also graph models that are built by anchor graphs [73]. An anchor graph is a hierarchical representation of a target graph. It is built with a small subset of the instances. This small subset is used to retain the similarities between all instances. In such a representation, the label matrix inversion is the main cost of training. To reduce the cost of the matrix inversion, the pruning of anchors' adjacency [74] is a common technique.

Performance improvements for deep neural networks can be achieved in two different ways. First, activation functions, such as Rectified Linear Unit (ReLU) [75] and its variants [76] [77], can be employed instead of the expensive functions, such as sigmoid and tanh, which use the exponential function. Other techniques, specifically for CNNs, involve depth-wise filter factorization [78] and group-wise convolutions [79].

### 4.3.2 Optimization Approximation

Optimization approximation is a family of techniques that are used to reduce the cost of the optimization related computations, i.e., gradient computations, for training. It is generally realized by computing the gradients with a small number

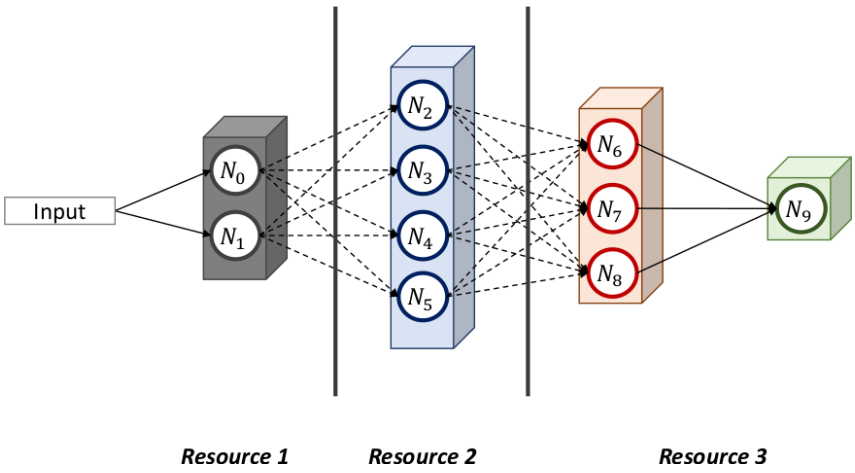
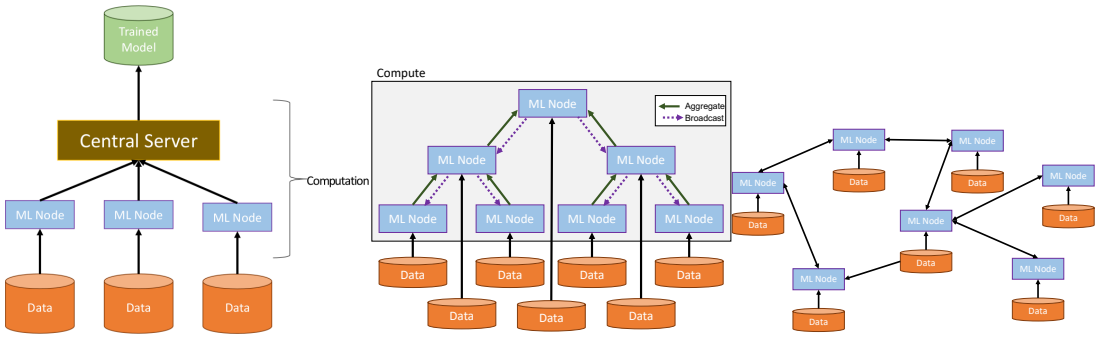



**Fig. 7**: Pipeline parallelism for a deep neural network.




(a) Centralized (b) Hierarchical (c) Distributed

**Fig. 8**: Different topologies of distributed ML.

of instances or parameters instead of all instances or parameters. Care has to be taken since such approximations can lead to longer convergence times, local extrema, or even non-convergence. Optimization approximation can be categorized based on the specific optimization algorithm that is being used: Mini-batch gradient descent, coordinate descent, and numerical integration based on Markov chain Monte Carlo. Table 2 shows the existing techniques.

Techniques that are used for mini-batch gradient descent approximations are adaptive sampling of mini-batches, adaptive learning rates, and the improvements in gradient approximations. Adaptive sampling [80] [81] for mini-batches takes the data distribution and gradient contributions into account rather than just using random batches of samples or making a gradual increase in the batch size [82]. Learning rates are also crucial in terms of achieving fast convergence [83]. Adaptive learning rates can boost the speed and quality of convergence [84]. Further adaptive adjustments are shown to be effective [85] [86]. Complementary to adaptive sampling or adaptive learning rates, reducing the variance of gradients and computing more accurate gradients are shown to be effective and efficient in achieving fast convergence. Such methods use average gradients or look-ahead corrections of gradients [87] [88]. In addition to the accurate first-order gradients, higher-order gradients may be needed due to ill-conditioning [89] [90]. Hessian matrices are estimated by the high-order gradients to make convergence possible [89].

Coordinate gradient descent are targeted at the problems where the instances are high dimensional, such as recommender systems [91] and natural language processing [92]. To speed up the

| Model Type | Techniques | Existing Work |
|---|---|---|
| Kernel-based Models | Sampling-based<br>Projection-based | [64, 65, 66, 67] |
| Tree-based Models | Rule sampling<br>Feature sampling | [68, 69, 52] |
| Graph-based Models | Sparse graph construction<br>Anchor graph based optimization | [70, 71, 72, 73, 74] |
| Deep Neural Network Models | Efficient activation functions<br>Filter factorization and grouping | [75, 76, 77, 78, 79] |

**Table 1**: Model simplifications for different ML models.

| Categories | Techniques | Existing Work |
|---|---|---|
| Mini-batch gradient descent | Adaptive sampling<br>Adaptive learning rates<br>Gradient corrections | [80], [81], [82]<br>[83] [84]<br>[85] [86]<br>[87], [88]<br>[89], [90] |
| Coordinate gradient descent | Rule sampling<br>Feature sampling | [91] [92]<br>[93] [94] [95]<br>[96] [94] [97] [98] |
| Bayesian optimization | Sparse graph construction<br>Anchor graph based optimization | [99], [100], [101] |

**Table 2**: Optimization approximation based techniques.

optimizations performed by coordinate gradient descent, a small number of parameters can be selected at each iteration. Random selection of parameters has shown to be effective [93] [94]. Parameter selection can also be based on the first and/or second-order gradients information [95] [96]. Another approach for speedup is to use extrapolation steps during the optimization phase [94]. If the optimization problem is non-convex, then studies such as [97] [98] present specific solutions. For instance, Li and Lin [97] propose an extended variant of accelerated proximal gradient method.

Finally, Bayesian optimization methods are commonly based on Markov chain Monte Carlo [99] [100]. Such methods employ stochastic mini-batches due to the high cost of the acceptance tests [101].

### 4.3.3 Communication Optimization Approaches

Optimizations to reduce communication costs constitute another option to those for computation. In these optimizations, compression of gradients is one of the two main ideas. Some studies compress each gradient component to just 1 bit [102]. Others map gradients to a discrete set of values [103] or sketch gradients into buckets and then encode them [104]. Some proposals only communicate gradients that are bigger than a certain threshold [105]. A combination of gradient compression and low-precision learning has been shown to further reduce the communication costs [106]. The other main idea for the optimization of communication is gradient delaying [107]. Ho et. al. explore the usage of gradient delays for stale synchronous parallel communications. Zheng et. al. [108] on the other hand compute approximate second-order gradients and overlap these computations with the delays to enhance the communication efficiency. Zhang, Choromanska, and LeCun [109] define an elastic relationship between the local and global model to avoid local minima as gradient transfers are delayed. Different than these studies, McMahan and Streeter [110] introduce communication optimizations for online learning.

Table 3 summarizes these techniques.

| Categories | Techniques | Existing Work |
|---|---|---|
| Communications | Gradient compression | [102], [103], [104] [105], [106] |
| | Gradient delay | [107], [108], [109], [110] |

**Table 3**: Communication optimization approaches.

## 4.4 Communication Topology

In a distributed ML system, the computing resources (clusters) can be structured in different ways. The types of topologies that the resources use can be categorized into three: centralized, hierarchical, and fully distributed (decentralized). Figure 8 depicts these topologies. Table 4 summarizes our discussion.

### 4.4.1 Centralized Topology

In this topology, the computation of the global model parameters, gradient averaging and communications with the distributed nodes/clients are performed at a central server. Every distributed client directly communicates with the central server and works with its local data only. A major disadvantage of a centralized topology is that the central server constitutes a single point of failure and a computational bottleneck. Advantages of a centralized topology are the ease of its implementation and inspection. Figure 8 (a) presents an example of this topology.

### 4.4.2 Hierarchical Topology

The computations and aggregation of the global model parameters are performed in a stage-wise and hierarchical way. Each child node only communicates with its parent. These topologies offer higher scalability than the centralized counterparts and easier manageability than the distributed counterparts. Figure 8 (b) depicts a hierarchical topology.

### 4.4.3 Fully Distributed Topology

Every participant maintains a local copy of the global model in a fully distributed topology. Participants directly communicate with each other. Compared to the centralized and hierarchical topologies, scalability is much higher and the single points of failure are eliminated. However, the implementation of these topologies is relatively more complex. Figure 8 (c) shows this topology.

## 4.5 Synchronization Models

Synchronization models are techniques to guide and perform synchronization between parallel computations and communications. These models seek to establish the best trade-off between fast updates and accurate models. To do fast updates, lower levels of synchronization are required. In comparison, to obtain accurate models, higher levels of synchronization are needed.

As far as ML is concerned, stochastic gradient descent is one of the most popular algorithms for the optimization during the training phase. As discussed below, variants of stochastic gradient descent have been implemented in accordance with the underlying synchronization model. Therefore, those variants constitute practical examples for the corresponding synchronization model.

### 4.5.1 Bulk Synchronous Parallel

It is a synchronization model [111] where synchronization happens between each computation and communication phase. Since this model is serializable by construction, the final output is guaranteed to be correct. However, when there are discrepancy between the progress of parallel workers, the faster workers have to wait for the slower ones. This can result in significant synchronization overhead.

### 4.5.2 Stale Synchronous Parallel

This synchronization model [107] allows the faster workers continue with their version of data for an additional but limited number of iterations to reduce the synchronization overheads due to the wait on the slower workers. While this can help reduce the overheads, data consistency and model convergence may become difficult to establish.

### 4.5.3 Approximate Synchronous Parallel

In this model, synchronization is sometimes omitted or delayed to reduce the overheads. However,

| Topology | Complexity | Scalability | Manageability | Single Point Failures | Latency |
|---|---|---|---|---|---|
| Centralized | Low | Low | High | Yes | Low |
| Hierarchical | Medium | Medium | Medium | Yes | Medium |
| Fully Distributed | High | High | Low | No | High |

**Table 4**: Comparison of different communication topologies.

the accuracy and consistency of a model may deteriorate if care is not taken. An advantage of approximate synchronicity is that when a parameter update is insignificant, the server can delay synchronization as much as possible. A disadvantage is that selecting which updates are significant or not is typically difficult to do. As an example of the application of this model, Gaia [112] is an approximate synchronous parallel ML system.

### 4.5.4 Asynchronous Parallel

This synchronization model omits all synchronizations among the workers. While these omissions may significantly reduce the computation time and communication overhead, asynchronous communications may cause ML models to produce incorrect outputs. To give an example application, HOGWILD algorithms [113] are developed based on asynchronous communications.

## 4.6 Existing Distributed Learning Frameworks

There are many ML frameworks that provide distributed ML algorithms and utilities. The most popular distributed implementations are Tensorflow [114, 115, 116], PyTorch [117, 118], MXNet [119, 120], Horovod [121], Baidu [122], Dianne [123], CNTK [124] and Theano [125]. Table 5 summarizes these frameworks. Other than the ML frameworks above, some general-purpose distributed computing libraries, such as Apache Spark [126] and Hadoop [127], also support distributed ML.

Tensorflow [114] is a free and open-source software library developed for ML and DL by Google. In fact, Tensorflow is the most popular library among the DL libraries. It supports distributed learning with several distribution strategies, such as mirrored, multi-worker and parameter server, that are either data or model parallel [115, 116]. The library provides efficient and scalable ML implementations for CPUs, multi-GPUs and mobile devices.

PyTorch [117] is another free and open-source framework based on the Torch Library developed by Meta. It is a popular framework for scientific research and provides automatic differentiation and dynamic computation graphs. It supports distributed learning mainly in two ways with torch.distributed package [118]. First, same as the Tensorflow mirrored strategy, PyTorch offers distributed data-parallel training which is based on the single-program and multiple-data paradigm. Second, for the cases that do not fit into data parallelism, PyTorch provides Remote Procedure Call (RPC) based distributed training. Examples of these types of distributed training are parameter server, pipeline parallelism, and reinforcement learning with multiple agents and observers.

MXNet [119] is an open-source DL framework for research prototyping and production. It offers data-parallel distributed learning with parameter servers. MXNet allows mixing both symbolic and imperative programming for computational efficiency and scalability. MXNet supports many programming languages such as C++, Python, R and Julia.

Horovod [121] is a distributed wrapper DL framework for TensorFlow, Keras, PyTorch, and Apache MXNet. Horovod is often easy to use because it only requires an addition of a small number of library calls to the source code. Horovod supports data, model and pipeline parallelisms.

Baidu [122] was started as an easy-to-use, efficient distributed DL platform. It supports large-scale ML and can train hundreds of machines in parallel with GPUs. Baidu offers various commercial solutions, such as machine translation, recommender systems, image classification and segmentation.

Dianne [123] is a distributed and ANNs-focused software framework based on OSGi which is a dynamic module system for Java. Dianne supports both model and data parallelisms and offers UI-based functionality.

| Frameworks | Pros | Cons | Parallelism |
|---|---|---|---|
| Tensorflow | Most popular.<br>Strong support by Google.<br>Efficient and scalable CPU,<br>multi-GPU,<br>mobile implementations.<br>Various training strategies:<br>Multi-worker, Parameter server... | Difficult to use API | Data, Model |
| PyTorch | Dynamic computation graph<br>Automatic differentiation<br>Support of remote procedure calls | No support for mobile | Data, Model,<br>Pipeline |
| MXNet | High scalability<br>Support of many languages:<br>C++, Python, Julia, R<br>Usage of symbolic<br>and imperative programming | Difficult to use API | Data |
| Horovod | Easy to use<br>Supports Tensorflow, Keras,<br>PyTorch, and MXNet | Lacks fault tolerance | Data Model<br>Pipeline |
| Baidu | Commercial ML and DL solutions | Limited scalability<br>No support for fault-tolerance | Data, Pipeline |
| Dianne | Java based development platform | No other languages | Data, Model |
| CNTK | Open-source<br>Efficient and high-performing | No longer actively developed<br>Limited mobile support | Data, Model |
| Theano | Open-source and cross-platform<br>Powerful numerical library | Discontinued | Data |

**Table 5**: Existing distributed learning platforms.

The Microsoft Cognitive Toolkit (CNTK) [124] is open-source software for commercial-grade DL. However, it is no longer actively developed. It supports distributed learning through parallel Stochastic Gradient Descent (SGD) algorithms. CNTK implements the following four parallel SGD algorithms: Data-parallel, block momentum, model averaging, and asynchronous data-parallel SGD.

Theano [125] was a popular open-source Python library to define, optimize and evaluate mathematical expressions. It has support for efficient multi-dimensional arrays. Developed by Universite de Montreal, it is no longer used widely. Theano supports data-parallel distributed learning by both synchronous and asynchronous training. It also supports multi-GPU multi-machine distributed training.

General-purpose distributed frameworks that are based on MapReduce programming model [128], such as Apache Spark [126] and Apache Hadoop [127], supports distributed ML algorithms, applications and utilities. Apache Spark is one of the most popular implementations of MapReduce. It includes MLlib [129] which is an open-source scalable distributed ML library. MLlib consists of widely-used ML algorithms and utilities for classification, regression, clustering, and dimensionality reduction tasks.

# 5 Federated Learning (FL)

In this section, we first introduce FL. We then present the existing aggregation algorithms in detail. After that, we discuss the security and privacy aspects of FL. We conclude this section by the available FL platforms and datasets.

## 5.1 Introduction to FL

FL [9] is a variant of ML where training a model is done by distributed clients that individually train local models. Once local models are trained, all

local model parameters are sent to a central server which then calculates the average of the parameters (weights) to compute an average model. This average model is then communicated back to the clients for subsequent local training. FL performs distributed training without sharing private client data.

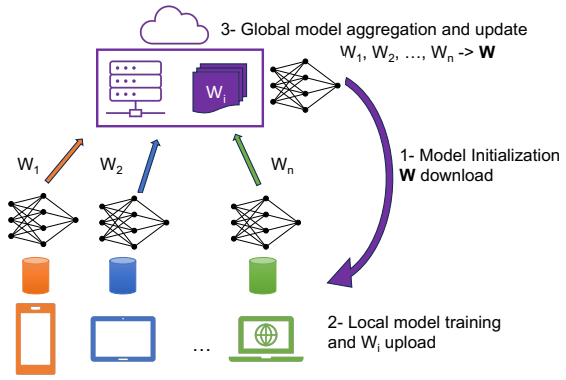



**Fig. 9**: Federated Learning Overview.

FL can be categorized based on how data partitioning is done. Horizontal FL [130] refers to the case where the clients share the same feature space but have different sample spaces. This is similar to data parallelism. An example of horizontal FL is wake-up voice recognition on smartphones. Users with different types of voices (different sample spaces) speak the same wake-up command (same feature space).

Vertical FL [130] takes place where the clients share the same sample space but have different feature spaces. As an example, the common customers (same sample space) of a bank and an e-commerce company (different feature spaces) join the training of an FL model for optimizing personal loans.

Finally, Federated Transfer Learning [131] refers to the case where both the sample and the feature spaces are different. Federated transfer learning transfers features from different feature spaces to the same representation to train a model with the data of different clients. An example is disease diagnosis by many different collaborating countries with multiple hospitals which have different patients (different sample spaces) with different medication tests (different feature spaces).

Distributed machine learning and FL have some fundamental differences. These are:

- While distributed machine learning's main goal is to minimize the computational costs and achieve high scalability, FL's main goal is to provide privacy and security for the user/client data. As a result, FL is designed such that user/client data is never shared.
- Distributed learning assumes that the user data is independent and identically distributed (i.i.d). On the other hand, FL assumes non-i.i.d because users typically have different data distributions and types.
- Distributed learning is performed based on aggregating client data, which is then distributed to different clients for training and inference. Contrarily, FL utilizes decentralized data. The client data is never shared and is never aggregated on a central server.

**Algorithm 1** Federated learning: client and server functions

$i \leftarrow isClient$ or $isServer$
$E \leftarrow totalEpochs$
$B \leftarrow totalNumBatches$
$\eta \leftarrow learningRate$
$w \leftarrow initialWeights$
**if** $i = isClient$ **then**
  **function** UpdateClientWeight(w,k)
    **for** epochs $e$ from 1 to $E$ **do**
      **for** batchs $b$ from 1 to $B$ **do**
        $w \leftarrow w - \eta \nabla l(w, b)$
      **end for**
    **end for**
    **return** w to the server
  **end function**
**else**
  **function** ServerUpdateWeight
    $t \leftarrow currentRoundID$
    **for** $k$ in sub batch of $K$ clients **do**
      the following is done in parallel
      $w_{t+1}^k \leftarrow UpdateClientWeight(k, w_t^k)$
    **end for**
    $w_{t+1} \leftarrow \frac{\sum_{k=1}^{K} w_{t+1}^k}{K}$
  **end function**
**end if**

We now discuss the very first FL algorithm, FedAvg, proposed by McMahan et. al. [9]. Algorithm 1 describes FedAvg. It shows the action taken by the server and clients during a round of FL. The clients train the model with their data. Once trained, the weights are sent to the server as described by the $UpdateClientWeight$ function on line 7. Once the server receives the weights from the clients, which is done in parallel, it averages out all the weights and sends the average weights back to each clients, as seen in the $ServerUpdateWeight$ function on line 16. Training is repeated if the data changes. This is to keep the weights updated.

## 5.2 FL Applications

FL has a wide range of applications across different domains and settings. Some of them are:

- Smartphones: FL has been used to develop ML applications for smartphones such as next-word prediction, and face and voice recognition.
- Healthcare: FL has been applied successfully for research problems in medical studies such as drug discovery and brain tumor segmentation.
- The internet of things (IoT): IoT is a network of digital or mechanical computing objects that have sensors, software, and other computing technologies. IoT exchanges data with other devices and systems over the internet to perform specific learning tasks. Applications of FL in IoT include autonomous driving and intrusion and anomaly detection.
- Finance: FL has been adopted to detect/identify financial crimes such as fraudulent loans and money laundering.

## 5.3 FL Aggregation Algorithms

In FL, due to data parallelism and horizontal FL, aggregation algorithms are needed to aggregate the models or gradients between the participants. As stated above, the very first aggregation algorithm, called Federated Averaging (FedAvg), was introduced by McMahan et. al. [9] who essentially kick-started FL itself. FedAvg computes the global model parameters by averaging the parameter updates of the participants. Once the global parameters are computed and updated, these parameters are communicated back to the participants. FedAvg is a straightforward algorithm however, it is biased toward the participants who have favorable network conditions.

Aggregation algorithms have been studied extensively for centralized topologies [132, 133]. To decrease the communication overheads, Liu et. al. propose the Federated Stochastic Block Coordinate Descent (FedBCD) algorithm [132] in which each participant makes multiple local updates before synchronizing with other participants. Differently, FedOpt [133] uses gradient compression to reduce communication overhead while sacrificing accuracy. Furthermore, for edge devices where computational resources are limited, algorithms such as FedGKT [134] are developed.

A significant objective in FL is to provide fairness. Fairness means that the clients equally contribute to the global model with respect to certain metrics. Researchers have proposed algorithms such as Stochastic Agnostic Federated Learning (SAFL) [135] and FedMGDA+ [136] to achieve fairness.

Adaptive FL and its impact on convergence and accuracy have been explored in various recent works. ADAGRAD [137] offers an adaptive approach to ML optimization compared to FedAvg. ADAGRAD and its variants dynamically choose server and client learning rates and momentum parameters during training. Mime Lite [138] is a closely related study where adaptive learning rates and momenta are reported to improve accuracy.

Some recent aggregation algorithms support heterogeneity of participant data. FedProx [139] is such an algorithm used for FL over heterogeneous data and resources. SCAFFOLD [140] is another algorithm that accounts for heterogeneous data while reducing the number of rounds to converge. FedAtt [141] accounts for the client contributions by attending to the importance of their model updates. The attention is quantified by the similarity between the server model and the client model in a layer-wise manner. FedNova [142] proposes a normalized averaging method as a way to avoid objective inconsistencies and to achieve fast convergence for highly heterogeneous clients.

Personalization is another important consideration in FL. There has been extensive research on personalized FL [143] [144]. Tan et. al. [145] offer a survey of the latest personalization techniques.

When the topology of the clients in FL is hierarchical, an aggregation algorithm needs to take the hierarchy into account. Numerous hierarchical aggregation algorithms have been proposed for such settings [146, 147]. Among hierarchical solutions, SPAHM [146] and PFNM [147] are Bayesian FL methods. Similarly, for decentralized topologies, decentralized algorithms have been developed [148, 149].

Considering fault-tolerance in FL, Krum [150] is an aggregation scheme that is reportedly resilient to Byzantine failures [151] where computing processes fail arbitrarily and failure symptoms are different for different observers. For these types of failures, more fault-tolerant FL studies are needed.

## 5.4 Security and Privacy in FL

The security of FL entails ensuring the triad of confidentiality, integrity and availability of its data and models, and particularly, data privacy. Privacy is defined as the protection of the raw data against the information leakage. In this section, we first summarize the attack types and then the defensive actions and methods existing in the FL literature [26].

### 5.4.1 Attacks

There are numerous attack types in FL. Poisoning attacks aim to tamper with and/or alter the data or the model. Data poisoning [156, 157, 158] refers to altering the features in the training data or generating false data to degrade the performance of a model on the unseen data. Model poisoning [159, 160, 161] refers to the modification of the model parameters and/or the fabrication of false weights that are communicated between the participants and the servers.

Backdoor attacks [177, 178] inject malicious instructions into the models while not impacting their expected performance. These attacks are non-transparent and notoriously difficult to detect.

Inference attacks [26, 162, 163] involve gaining knowledge of the sensitive information of the participants, the training data or the model through the communications occurring during training or inference. Membership inference attacks aim to learn if a sample has been used as a training instance. Property inference attacks aim to learn the meta-characteristics of the training data. Class representative inference attacks aim to learn representative samples of a target class.

Generative Adversarial Networks (GANs) based attacks [166, 167, 168] are used to launch poisoning attacks where GANs generate the altered or false data and/or model parameters.

There are many other attack types [179, 180], such free-riders [26, 163] and Eavesdropping [26, 163].

### 5.4.2 Defenses

The most commonly used attack defense mechanisms can be categorized by the usage of trusted execution environments [169, 170], homomorphic encryption [164, 165], differential privacy [28, 154, 155], and possibly some combinations of them. There are many other techniques which are based on GANs [181], anomaly detection [176], secure multi-party computation [171], data anonymization [175], and blockchains [27, 172].

A trusted execution environment [169, 170] is an (hardware/software) architecture where the program execution is secured and information leakage is not possible. Such architectures use specialized designs to prevent unauthorized accesses as well as privacy violations in FL [169, 170].

Homomorphic encryption [182] is a certain type of encryption in which the decryption of the results of the computations performed on the encrypted data is the same as the result of the same computations performed on the unencrypted data. Homomorphic encryption has various levels depending on the whether addition and/or multiplication is supported. It has been adapted for data privacy in FL [164, 165].

Differential privacy [152, 153] is a technique for achieving data privacy by adding noise to raw data. It is commonly used in FL [28, 154, 155].

Table 6 reviews attacks and defenses in FL. In addition, Table 7 compares the defense mechanisms in terms of the strength of the protection, the computational and communication efficiency, robustness, scalability, and generalizability of a mechanism.

## 5.5 Existing FL Frameworks

The most widely used FL frameworks are TensorFlow Federated [183, 184], IBM Federated Learning [185], NVIDIA FLARE [186], FedML

| Defense Type | Addressed Attacks | Potential Negative Effects |
|---|---|---|
| Differential privacy [28, 152, 153, 154, 155] | Data Poisoning [156, 157, 158]<br>Model Poisoning [159, 160, 161]<br>Inference attacks [26, 162, 163] | Decreased model utility |
| Homomorphic encryption [164, 165] | Inference attacks [26, 162, 163]<br>GAN-based attacks [166, 167, 168] | High computational costs |
| Trusted execution environments [169, 170] | Inference attacks [26, 162, 163]<br>Model Poisoning [159, 160, 161] | Specialized hardware |
| Secure Multi-party Computation [171] | GAN-based attacks [166, 167, 168]<br>Inference attacks [26, 162, 163]<br>Eavesdropping [26, 163] | High computational costs |
| Blockchain [27, 172] | Blockchain attacks [173, 174] | High resource costs |
| Data anonymization [175] | GAN-based attacks [166, 167, 168]<br>Inference attacks [26, 162, 163] | Decreased data usability |
| Anomaly detection [176] | Data Poisoning [156, 157, 158]<br>Model Poisoning [159, 160, 161]<br>Free-riders [26, 163] | Detection latency |

**Table 6**: Attacks and defenses in FL.

| Defense Type | Protection | Efficiency | Robustness | Scalability | Generalizability |
|---|---|---|---|---|---|
| Differential privacy [28, 152, 153, 154, 155] | High | High | High | High | High |
| Homomorphic encryption [182] | High | Low | High | Low | High |
| Trusted execution environments [169, 170] | Medium | High | Medium | Low | High |
| Secure multi-party computations [171] | High | Medium | High | Low | Medium |
| Blockchain [27, 172] | High | Low | Medium | High | Medium |
| Data anonymization [175] | Medium | Medium | Low | High | High |
| Anomaly detection [176] | Medium | High | Medium | High | Low |

**Table 7**: Comparison of the defense mechanisms in terms of the strength of the protection, the computational and communication efficiency, robustness, scalability, and generalizability of a mechanism.

[187], Federated AI Technology Enabler (FATE) [188], PySyft [189], and Open Federated Learning (OpenFL) [190]. Table 8 summarizes the existing FL frameworks.

TensorFlow Federated [183] (and Keras Federated [184]) is an open-source framework for FL by Google. It enables researchers to simulate FL algorithms. FedAvg, FedProx, FedSGD, and Mime Lite are some of the FL aggregation algorithms that are readily available. TensorFlow Federated supports data and model parallelisms. It provides differential privacy as a privacy measure. TensorFlow Federated has two main APIs. FL API offers built-in algorithms. FL Core API offers a set of lower-level functionalities for new algorithms to be implemented.

IBM Federated Learning [185] provides support for FL and DL models written in Keras, PyTorch and TensorFlow. FedAvg, SPAHM, PFNM, and Krum are among the available aggregation algorithms. IBM FL supports data and model parallelism. In addition, differential privacy, secure multi-party computation and homomorphic encryption are available defenses for ensuring privacy and security. IBM FL also offers the implementations of several topologies and communication protocols.

| Frameworks | Aggregation Algorithm | Parallelism | Privacy and Security |
|---|---|---|---|
| TensorFlow Federated<br>Keras Federated | FedAvg, FedProx,<br>FedSGD, Mime Lite | Data, Model | Differential privacy |
| IBM Federated | FedAvg, SPAHM,<br>PFNM, Krum | Data, Model | Differential privacy<br>Secure multi-party computation<br>Homomorphic encryptions |
| NVIDIA FLARE | FedAvg, FedProx,<br>SCAFFOLD | Data | Differential privacy<br>Homomorphic encryption |
| FedML | FedAvg, FedOpt,<br>FedNova, FedGKT | Data, Model | Differential privacy<br>Cryptography<br>Coding approaches |
| FATE | FedAvg | Data, Pipeline | Homomorphic encryption<br>RSA |
| PySyft | FedAvg, FedProx,<br>FedSGD | Data, Model | Differential privacy<br>Homomorphic encryption |
| OpenFL | FedAvg,<br>FedADAGRAD | Data | Trusted execution environments<br>RSA<br>Differential privacy |

**Table 8**: Existing FL platforms.

NVIDIA FLARE (Federated Learning Application Runtime Environment) [186] is a modular open-source software development kit (SDK) for FL which offers secure and privacy-preserving distributed learning. FLARE provides FL algorithms such as FedAvg, FedProx and SCAFFOLD. It offers differential privacy and homomorphic encryption. FLARE SDK has several components, such as a simulator for prototyping, secure management tools for provisioning and deployment and an API for extensions.

FedML [187] framework offers a wide-range of cross-platform FL capabilities including natural language processing, computer vision, and GNNs. FedAvg, FedOpt, FedNova and FedGKT are the supported FL algorithms. FedML offers defense mechanisms such as differential privacy, cryptography routines, and several coding methods. It supports data and model parallel distributed learning. FedML models can be trained and deployed at the edge or on the cloud.

FATE [188] is an open-source platform initiated by WeBank, a bank based in Shenzhen, China. It provides a diverse set of FL algorithms, such as tree-based algorithms, DL, and transfer learning. It offers a set of modules consisting of an ML algorithms library, a high-performance serving system, an end-to-end pipeline system, a multi-party communication network system, and a module for cloud technologies. FATE provides homomorphic encryption and RSA for secure and privacy preserving training. FATE supports data and pipeline parallelisms.

PySyft [189] is an open-source multi-language library that provides secure and private DL and FL in Python for frameworks such as PyTorch, Tensorflow and Keras. It supports differential privacy and homomorphic encryption. FedAvg, FedProx and FedSGD are among the available aggregation algorithms. Training can be data or model parallel.

OpenFL [190] is an open-source Python framework originally developed by Intel Labs. It provides a set of workflows for the researchers to experiment with FL. FedAvg and ADAGRAD algorithms are built-in. OpenFL's capabilities include trusted execution environments, RSA, differential privacy.

## 5.6 FL Datasets

As FL research progresses, new datasets are being built. One of the most well-known datasets for FL is the LEAF [191]. It is a suite of open-source federated datasets. There are a total of six different datasets. One of the datasets, called FEMNIST, is built for image classification. Sentiment140, which consists of Tweets, is a dataset for sentiment analysis. Shakespeare is a text dataset of

Shakespeare Dialogues which is used for next character prediction. Celeba is an image classification dataset of celebrity images. There is a synthetic classification dataset which is generated for the FL models that are device-dependant. Lastly, the Reddit comments dataset is used for next word prediction.

TensorFlow Federated [192] offers several datasets to support FL simulations. While some of its datasets are the same as those of LEAF, there are also different datasets, such as the federated CIFAR-100 dataset, the FLAIR dataset, and the federated Google Landmark v2 dataset.

Street Dataset [193] is a real-world image dataset. It contains images generated from street cameras. A total of seven object categories annotated with bounding boxes. This dataset is built for object detection tasks.

CC-19 [194] is a new dataset related to the latest family of coronavirus (COVID-19). It contains the Computed Tomography (CT) scan of subjects and is built for image classification.

FedTADBench [195] offers three different datasets to evaluate time series anomaly detection algorithms.

# 6 Open Questions and Challenges

In this section, we summarize the challenges that ML and FL face. We only present major problems. This is because there is a large number of open problems, and we choose to keep our presentation concise and focused.

## 6.1 Challenges for Parallel and Distributed ML

The major challenges with parallel and distributed ML are related to performance, fault-tolerance, security and privacy [7, 196, 197].

Typically, in distributed and parallel training, additional resources are used to decrease wall-clock time [198]. Such additional resources can be multiple machines, multiple GPUs and high-end communication networks. As a result, the decrease in wall-clock time may not compensate for the additional resources or their energy consumption. Therefore, research studies, such as [16], are needed to investigate this trade-off with different applications and system architectures.

Distributed and parallel ML platforms, especially those executed on high-performance computing systems, often consider fault-tolerance as a second-class concern. However, given the sizes of the latest large-scale computing systems, failures are common; not rare [199]. As a result, efficient checkpointing and/or replication solutions [199] are needed to recover from errors and to limit the amount of lost computation due to a failure.

Ensuring security and privacy for distributed and parallel ML has consistently been a serious concern [196]. While FL was devised for the privacy of user data, there have been many novel types of attacks [179, 180]. These attacks include adversarial [168], poisoning, evasion, backdoor, and integrity attacks [180, 197]. As such attacks get sophisticated, so must their defenses. Moreover, the systematic deployment of the defenses to the physical systems as well as the evaluation of these deployments have not studied well [197]. Furthermore, there is a lack of the rigorous efficiency and efficacy studies of attack defense mechanisms. As a result of these issues, security and privacy for ML remain an open problem.

## 6.2 Challenges for FL

The main challenges in FL are two-fold [200]: explainability and interpretability, and federated GNNs. Explainability and interpretability refer to the understanding of the contributions of the clients or the data features. For instance, Shapley values are proposed [201] to quantify the impact of the features on the model output. Zheng et. al. propose a quantified ranking of features [202]. Similarly, there are studies [203] targeting vertical FL. Several works introduce tailored measures of interpretability such as [204] defining a measure based on the gradients. However, in general, the problem of explainability and interpretability remains open because i) ensuring privacy while building explainable models is not trivial, ii) the aggregation of the local parameters obscures interpretability, iii) there is a lack of datasets that are not composed of images or text, and iv) there is a lack of a general framework for explainable federated models.

Research for FL with GNNs [205, 206, 207] has recently started. For instance, FedGraphNN [205] provides an FL benchmark system to evaluate various graph models, algorithms and datasets.

Another example is GraphFL [207] which is designed to classify nodes on graphs. However, many questions are still waiting to be solved, such as the protection against malicious attacks, interpretability, lack of modern graph neural frameworks for FL [208].

# 7 Conclusions

In this work, we provided a review of modern large-scale, parallel and distributed ML: the state-of-the-art algorithms, optimization methods, types of parallelisms, communication topologies, synchronization models, and the existing frameworks. Moreover, we reviewed FL. We discussed various aggregation algorithms in FL. In addition, we reviewed the security and privacy aspects including various types of attacks and defense mechanisms. Moreover, we explored the existing FL frameworks and datasets. We concluded our study with the open research problems and challenges in large-scale distributed ML and FL. The major challenges are typically related to performance, security, privacy, explainability, portability, and fault-tolerance.

# Acknowledgment

This work was supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation" project. The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

# References

[1] Shi Dong, Ping Wang, and Khushnood Abbas. A survey on deep learning and its applications. *Computer Science Review*, 40:100379, 2021.

[2] Iqbal H Sarker. Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2(6):420, 2021.

[3] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey of machine learning accelerators. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–12, 2020.

[4] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy K̃epner. Survey and benchmarking of machine learning accelerators. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, 2019.

[5] Sathwika Bavikadi, Abhijitt Dhavlle, Amlan Ganguly, Anand Haridass, Hagar Hendy, Cory Merkel, Vijay Janapa Reddi, Purab Ranjan Sutradhar, Arun Joseph, and Sai Manoj Pudukotai Dinakarrao. A survey on machine learning accelerators and evolutionary hardware platforms. *IEEE Design & Test*, 39(3):91–116, 2022.

[6] Meng Wang, Weijie Fu, Xiangnan He, Shijie Hao, and Xindong Wu. A survey on large-scale machine learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(6):2574–2594, 2022.

[7] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. A survey on distributed machine learning. *ACM Comput. Surv.*, 53(2), mar 2020.

[8] Lucy Ellen Lwakatare, Aiswarya Raj, Ivica Crnkovic, Jan Bosch, and Helena Holmström Olsson. Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions. *Information and Software Technology*, 127:106368, 2020.

[9] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017.

[10] Ji Liu, Jizhou Huang, Yang Zhou, Xuhong Li, Shilei Ji, Haoyi Xiong, and Dejing Dou. From distributed machine learning to federated learning: A survey. *Knowl. Inf. Syst.*, 64(4):885–917, apr 2022.

[11] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

[12] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3347–3366, 2023.

[13] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021.

[14] Shaveta Dargan, Munish Kumar, Maruthi Rohit Ayyagari, and Gulshan Kumar. A survey of deep learning and its applications: a new paradigm to machine learning. *Archives of Computational Methods in Engineering*, 27:1071–1092, 2020.

[15] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021.

[16] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Comput. Surv.*, 52(4), aug 2019.

[17] Yuhao Zhang, Frank McQuillan, Nandish Jayaram, Nikhil Kak, Ekta Khanna, Orhan Kislal, Domino Valdano, and Arun Kumar. Distributed deep learning on data systems: A comparative analysis of approaches. *Proc. VLDB Endow.*, 14(10):1769–1782, jun 2021.

[18] Matthias Langer, Zhen He, Wenny Rahayu, and Yanbo Xue. Distributed training of deep learning models: A taxonomic perspective. *IEEE Transactions on Parallel and Distributed Systems*, 31(12):2802–2818, 2020.

[19] Ruben Mayer and Hans-Arno Jacobsen. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *ACM Comput. Surv.*, 53(1), feb 2020.

[20] Omar Abdel Wahab, Azzam Mourad, Hadi Otrok, and Tarik Taleb. Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems. *IEEE Communications Surveys & Tutorials*, 23(2):1342–1397, 2021.

[21] Yingxia Shao, Hongzheng Li, Xizhi Gu, Hongbo Yin, Yawen Li, Xupeng Miao, Wentao Zhang, Bin Cui, and Lei Chen. Distributed graph neural network training: A survey, 2022.

[22] Haozhao Wang, Zhihao Qu, Qihua Zhou, Haobo Zhang, Boyuan Luo, Wenchao Xu, Song Guo, and Ruixuan Li. A comprehensive survey on training acceleration for large machine learning models in iot. *IEEE Internet of Things Journal*, 9(2):939–963, 2022.

[23] Shuyan Hu, Xiaojing Chen, Wei Ni, Ekram Hossain, and Xin Wang. Distributed machine learning for wireless communication networks: Techniques, architectures, and applications. *IEEE Communications Surveys & Tutorials*, 23(3):1458–1493, 2021.

[24] Omar Nassef, Wenting Sun, Hakimeh Purmehdi, Mallik Tatipamula, and Toktam Mahmoodi. A survey: Distributed machine learning for 5g and beyond. *Comput. Netw.*, 207(C), apr 2022.

[25] Yahao Ding, Zhaohui Yang, Quoc-Viet Pham, Zhaoyang Zhang, and Mohammad Shikh-Bahaei. Distributed machine learning for uav swarms: Computing, sensing, and semantics, 2023.

[26] Viraaji Mothukuri, Reza M. Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021.

[27] Youyang Qu, Md Palash Uddin, Chenquan Gan, Yong Xiang, Longxiang Gao, and John Yearwood. Blockchain-enabled federated learning: A survey. *ACM Computing Surveys*, 55(4):1–35, 2022.

[28] Ahmed El Ouadrhiri and Ahmed Abdelhadi. Differential privacy for deep and federated learning: A survey. *IEEE Access*, 10:22359–22380, 2022.

[29] Steven CH Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *Neurocomputing*, 459:249–289, 2021.

[30] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[31] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial intelligence review*, 53(8):5455–5516, 2020.

[32] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[33] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[35] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[36] Marc M Van Hulle. Self-organizing maps. *Handbook of natural computing*, 1:585–622, 2012.

[37] Geoffrey E Hinton, Terrence J Sejnowski, and David H Ackley. *Boltzmann machines: Constraint satisfaction networks that learn.* Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA, 1984.

[38] Geoffrey E Hinton, Terrence J Sejnowski, et al. Learning and relearning in boltzmann machines. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1(282-317):2, 1986.

[39] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

[40] John J Hopfield. Hopfield network. *Scholarpedia*, 2(5):1977, 2007.

[41] Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

[42] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *AI Open*, 2022.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[44] OpenAI. Gpt-4 technical report, Accessed on 03-29-2023.

[45] OpenAI. Chatgpt, Accessed on 03-29-2023.

[46] Sholom M Weiss and Nitin Indurkhya. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research*, 3:383–403, 1995.

[47] Akbar Telikani, Amirhessam Tahmassebi, Wolfgang Banzhaf, and Amir H. Gandomi. Evolutionary machine learning: A survey. *ACM Comput. Surv.*, 54(8), oct 2021.

[48] Rubayyi Alghamdi and Khalid Alfalqi. A survey of topic modeling in text mining. *Int. J. Adv. Comput. Sci. Appl.(IJACSA)*, 6(1), 2015.

[49] Pooja Kherwa and Poonam Bansal. Topic modeling: A comprehensive review. *EAI Endorsed Transactions on Scalable Information Systems*, 7(24), 7 2019.

[50] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[51] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2016.

[52] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[53] Sören Sonnenburg, Gunnar Ratsch, Sebastian Henschel, Christian Widmer, Jonas Behr, Alexander Zien, Fabio de Bona, Alexander Binder, Christian Gehl, and Vojtech Franc. The shogun machine learning toolbox. *Journal of Machine Learning Research*, 11(60):1799–1802, 2010.

[54] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[55] Google. Google cloud, Accessed on 03-27-2023.

[56] Microsoft. Microsoft azure, Accessed on 03-27-2023.

[57] Amazon. Machine learning on aws, Accessed on 03-27-2023.

[58] IBM Watson. Ibm watson assistant, Accessed on 03-27-2023.

[59] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.

[60] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12, jun 2017.

[61] Rengan Xu, Frank Han, and Quy Ta. Deep learning at scale on nvidia v100 accelerators. In *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 23–32, 2018.

[62] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. Dissecting the graphcore IPU architecture via microbenchmarking. *CoRR*, abs/1912.03413, 2019.

[63] Murali Emani, Venkatram Vishwanath, Corey Adams, Michael E. Papka, Rick Stevens, Laura Florescu, Sumti Jairath, William Liu, Tejas Nama, and Arvind Sujeeth. Accelerating scientific applications with sambanova reconfigurable dataflow architecture. *Computing in Science & Engineering*, 23(2):114–119, 2021.

[64] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling methods for the nyström method. *The Journal of Machine Learning Research*, 13(1):981–1006, 2012.

[65] Djallel Bouneffouf and Inanc Birol. Sampling with minimum sum of squared similarities for nystrom-based large scale spectral clustering. In *IJCAI*, pages 2313–2319, 2015.

[66] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30(1):47–68, 2011.

[67] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[68] Jia Deng, Sanjeev Satheesh, Alexander Berg, and Fei Li. Fast and balanced: Efficient label tree learning for large scale object recognition. *Advances in Neural Information Processing Systems*, 24, 2011.

[69] Yael Ben-Haim and Elad Tom-Tov. A streaming parallel decision tree algorithm. *Journal of Machine Learning Research*, 11(2), 2010.

[70] Wei Liu, Junfeng He, and Shih-Fu Chang. Large graph construction for scalable semi-supervised learning. In *Proceedings of the*

*27th international conference on machine learning (ICML-10)*, pages 679–686. Citeseer, 2010.

[71] Yan-Ming Zhang, Kaizhu Huang, Guanggang Geng, and Cheng-Lin Liu. Fast k nn graph construction with locality sensitive hashing. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part II 13*, pages 660–674. Springer, 2013.

[72] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):769–790, 2017.

[73] Meng Wang, Weijie Fu, Shijie Hao, Hengchang Liu, and Xindong Wu. Learning on big graph: Label inference and regularization with anchor hierarchy. *IEEE transactions on knowledge and data engineering*, 29(5):1101–1114, 2017.

[74] Meng Wang, Weijie Fu, Shijie Hao, Dacheng Tao, and Xindong Wu. Scalable semi-supervised learning by efficient anchor graph regularization. *IEEE Transactions on Knowledge and Data Engineering*, 28(7):1864–1877, 2016.

[75] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[76] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013.

[77] Shan Sung Liew, Mohamed Khalil-Hani, and Rabia Bakhteri. Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems. *Neurocomputing*, 216:718–734, 2016.

[78] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[79] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[80] Siddharth Gopal. Adaptive sampling for sgd by exploiting side information. In *International Conference on Machine Learning*, pages 364–372. PMLR, 2016.

[81] Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron Courville, and Yoshua Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

[82] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[83] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *Neural Networks: Tricks of the Trade: Second Edition*, pages 437–478, 2012.

[84] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[85] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[86] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[87] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[88] Yu Nesterov. Gradient methods for minimizing composite functions. *Mathematical programming*, 140(1):125–161, 2013.

[89] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.

[90] Quoc V Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th*

*International Conference on International Conference on Machine Learning*, pages 265–272, 2011.

[91] Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. A generic coordinate descent framework for learning from implicit feedback. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1341–1350, 2017.

[92] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012.

[93] Julie Nutini, Mark Schmidt, Issam Laradji, Michael Friedlander, and Hoyt Koepke. Coordinate descent converges faster with the gauss-southwell rule than random selection. In *International Conference on Machine Learning*, pages 1632–1641. PMLR, 2015.

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

[95] Hao-Jun Michael Shi, Shenyinying Tu, Yangyang Xu, and Wotao Yin. A primer on coordinate descent algorithms. *arXiv preprint arXiv:1610.00040*, 2016.

[96] Sangwoon Yun and Kim-Chuan Toh. A coordinate gradient descent method for l1-regularized convex minimization. *Computational Optimization and Applications*, 48(2):273–307, 2011.

[97] Huan Li and Zhouchen Lin. Accelerated proximal gradient methods for nonconvex programming. *Advances in neural information processing systems*, 28, 2015.

[98] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[99] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

[100] Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(suppl_1):5228–5235, 2004.

[101] Sungjin Ahn, Anoop Korattikara, and Max Welling. Bayesian posterior sampling via stochastic gradient fisher scoring. *arXiv preprint arXiv:1206.6380*, 2012.

[102] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth annual conference of the international speech communication association*, 2014.

[103] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

[104] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *Advances in neural information processing systems*, 30, 2017.

[105] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[106] Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. Zipml: Training linear models with end-to-end low precision, and a little bit of deep learning. In *International Conference on Machine Learning*, pages 4035–4043. PMLR, 2017.

[107] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. *Advances in neural information processing systems*, 26, 2013.

[108] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *International Conference on Machine Learning*, pages 4120–4129. PMLR, 2017.

[109] Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic

averaging sgd. *Advances in neural information processing systems*, 28, 2015.

[110] Brendan McMahan and Matthew Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. *Advances in Neural Information Processing Systems*, 27, 2014.

[111] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[112] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. Gaia: Geo-distributed machine learning approaching lan speeds. In *NSDI*, pages 629–647, 2017.

[113] Christopher De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild!-style algorithms, 2015.

[114] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[115] Tensorflow. Distributed training with tensorflow. Available at https://www.tensorflow.org/guide/distributed_training, Last accessed 09.01.2022.

[116] Keras. Distributed training with keras. Available at https://www.tensorflow.org/tutorials/distribute/keras, Last accessed 09.01.2022.

[117] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[118] PyTorch. Pytorch distributed overview. Available at https://pytorch.org/tutorials/beginner/dist_overview.html, Last accessed 09.01.2022.

[119] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[120] Amith R Mamidala, Georgios Kollias, Chris Ward, and Fausto Artico. Mxnet-mpi: Embedding mpi parallelism in parameter server task model for scaling deep learning, 2018.

[121] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799, 2018.

[122] Andrew Gibiansky. Bringing hpc techniques to deep learning. *Baidu Research, Tech. Rep.*, 2017.

[123] Elias De Coninck, Steven Bohez, Sam Leroux, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. Dianne: a modular framework for designing, training and deploying deep neural networks on heterogeneous distributed infrastructure. *Journal of Systems and Software*, 141:52–65, 2018.

[124] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 2135, New York, NY, USA, 2016. Association for Computing Machinery.

[125] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David

Warde-farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proceedings of the 9th Python in Science Conference*, pages 3–10, 2010.

[126] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[127] Apache Software Foundation. Hadoop.

[128] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[129] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. Mllib: Machine learning in apache spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[130] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[131] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. A secure federated transfer learning framework. *IEEE Intelligent Systems*, 35(4):70–82, 2020.

[132] Yang Liu, Xinwei Zhang, Yan Kang, Liping Li, Tianjian Chen, Mingyi Hong, and Qiang Yang. Fedbcd: A communication-efficient collaborative learning framework for distributed features. *IEEE Transactions on Signal Processing*, 70:4277–4290, 2022.

[133] Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. Fedopt: Towards communication efficiency and privacy preservation in federated learning. *Applied Sciences*, 10(8):2864, 2020.

[134] Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. *Advances in Neural Information Processing Systems*, 33:14068–14080, 2020.

[135] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning, 2019.

[136] Zeou Hu, Kiarash Shaloudegi, Guojun Zhang, and Yaoliang Yu. Federated learning meets multi-objective optimization, 2020.

[137] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[138] Sai Praneeth Karimireddy, Martin Jaggi, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Breaking the centralized barrier for cross-device federated learning. *Advances in Neural Information Processing Systems*, 34:28663–28676, 2021.

[139] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2018.

[140] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5132–5143. PMLR, 13–18 Jul 2020.

[141] Shaoxiong Ji, Shirui Pan, Guodong Long, Xue Li, Jing Jiang, and Zi Huang. Learning private neural language modeling with attentive aggregation. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[142] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems*, 33:7611–7623, 2020.

[143] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461*, 2020.

[144] Aviv Shamsian, Aviv Navon, Ethan Fetaya, and Gal Chechik. Personalized federated

learning using hypernetworks. In *International Conference on Machine Learning*, pages 9489–9502. PMLR, 2021.

[145] Alysa Ziying Tan, Han Yu, Lizhen Cui, and Qiang Yang. Towards personalized federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[146] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, and Nghia Hoang. Statistical model aggregation via parameter matching. *Advances in neural information processing systems*, 32, 2019.

[147] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *International conference on machine learning*, pages 7252–7261. PMLR, 2019.

[148] István Hegedűs, Gábor Danner, and Márk Jelasity. Decentralized learning works: An empirical comparison of gossip learning and federated learning. *Journal of Parallel and Distributed Computing*, 148:109–124, 2021.

[149] Hao Ye, Le Liang, and Geoffrey Ye Li. Decentralized federated learning with unreliable communications. *IEEE Journal of Selected Topics in Signal Processing*, 16(3):487–500, 2022.

[150] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in neural information processing systems*, 30, 2017.

[151] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the works of leslie lamport*, pages 203–226. ACM New York, NY, USA, 2019.

[152] Cynthia Dwork. Differential privacy. In *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33*, pages 1–12. Springer, 2006.

[153] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[154] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

[155] Stacey Truex, Ling Liu, Ka-Ho Chow, Mehmet Emre Gursoy, and Wenqi Wei. Ldpfed: Federated learning with local differential privacy. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, pages 61–66, 2020.

[156] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25*, pages 480–501. Springer, 2020.

[157] Florian Nuding and Rudolf Mayer. Data poisoning in sequential and parallel federated learning. In *Proceedings of the 2022 ACM on International Workshop on Security and Privacy Analytics*, IWSPA '22, page 24–34, 2022.

[158] Gan Sun, Yang Cong, Jiahua Dong, Qiang Wang, Lingjuan Lyu, and Ji Liu. Data poisoning attacks on federated machine learning. *IEEE Internet of Things Journal*, 9(13):11365–11375, 2022.

[159] Ashwinee Panda, Saeed Mahloujifar, Arjun Nitin Bhagoji, Supriyo Chakraborty, and Prateek Mittal. Sparsefed: Mitigating model poisoning attacks in federated learning with sparsification. In *International Conference on Artificial Intelligence and Statistics*, pages 7587–7624. PMLR, 2022.

[160] Xiaoyu Cao and Neil Zhenqiang Gong. Mpaf: Model poisoning attacks to federated learning based on fake clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3396–3404, 2022.

[161] Zhuoran Ma, Jianfeng Ma, Yinbin Miao, Yingjiu Li, and Robert H. Deng. Shieldfl: Mitigating model poisoning attacks in privacy-preserving federated learning. *IEEE Transactions on Information Forensics and*

*Security*, 17:1639–1654, 2022.

[162] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, pages 739–753. IEEE, 2019.

[163] Pengrui Liu, Xiangrui Xu, and Wei Wang. Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives. *Cybersecurity*, 5(1):1–19, 2022.

[164] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 2020)*, 2020.

[165] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[166] Poongodi Manoharan, Ranjan Walia, Celestine Iwendi, Tariq Ahamed Ahanger, ST Suganthi, MM Kamruzzaman, Sami Bourouis, Wajdi Alhakami, and Mounir Hamdi. Svm-based generative adverserial networks for federated learning and edge computing attack model and outpoising. *Expert Systems*, page e13072, 2022.

[167] Jiale Zhang, Bing Chen, Xiang Cheng, Huynh Thi Thanh Binh, and Shui Yu. Poisongan: Generative poisoning attacks against federated learning in edge computing systems. *IEEE Internet of Things Journal*, 8(5):3310–3322, 2020.

[168] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. Adversarial machine learning attacks and defense methods in the cyber security domain. *ACM Computing Surveys (CSUR)*, 54(5):1–36, 2021.

[169] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. Ppfl: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th annual international conference on mobile systems, applications, and services*, pages 94–108, 2021.

[170] Yu Chen, Fang Luo, Tong Li, Tao Xiang, Zheli Liu, and Jin Li. A training-integrity privacy-preserving federated learning scheme with trusted execution environment. *Information Sciences*, 522:69–79, 2020.

[171] Vaikkunth Mugunthan, Antigoni Polychroniadou, David Byrd, and Tucker Hybinette Balch. Smpai: Secure multi-party computation for federated learning. In *Proceedings of the NeurIPS 2019 Workshop on Robust AI in Financial Services*, 2019.

[172] Yunlong Lu, Xiaohong Huang, Yueyue Dai, Sabita Maharjan, and Yan Zhang. Blockchain and federated learning for privacy-preserved data sharing in industrial iot. *IEEE Transactions on Industrial Informatics*, 16(6):4177–4186, 2019.

[173] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and David Mohaisen. Exploring the attack surface of blockchain: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):1977–2008, 2020.

[174] Yourong Chen, Hao Chen, Yang Zhang, Meng Han, Madhuri Siddula, and Zhipeng Cai. A survey on blockchain systems: Attacks, defenses, and privacy preservation. *High-Confidence Computing*, 2(2):100048, 2022.

[175] Olivia Choudhury, Aris Gkoulalas-Divanis, Theodoros Salonidis, Issa Sylla, Yoonyoung Park, Grace Hsu, and Amar Das. Anonymizing data for privacy-preserving federated learning. *arXiv preprint arXiv:2002.09096*, 2020.

[176] Suyi Li, Yong Cheng, Yang Liu, Wei Wang, and Tianjian Chen. Abnormal client behavior detection in federated learning. *arXiv preprint arXiv:1910.09933*, 2019.

[177] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[178] Xueluan Gong, Yanjiao Chen, Qian Wang, and Weihan Kong. Backdoor attacks and

defenses in federated learning: State-of-the-art, taxonomy, and future directions. *IEEE Wireless Communications*, 2022.

[179] Maria Rigaki and Sebastian Garcia. A survey of privacy attacks in machine learning. *arXiv preprint arXiv:2007.07646*, 2020.

[180] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1563–1580, 2022.

[181] Ying Zhao, Junjun Chen, Jiale Zhang, Di Wu, Jian Teng, and Shui Yu. Pdgan: A novel poisoning defense method in federated learning using generative adversarial network. In *Algorithms and Architectures for Parallel Processing: 19th International Conference, ICA3PP*, page 595–609, 2019.

[182] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for non-specialists. *EURASIP Journal on Information Security*, 2007:1–10, 2007.

[183] Tensorflow. Federated learning. Available at https://www.tensorflow.org/federated/federated_learning, Last accessed 09.01.2022.

[184] Keras. Federated learning for image classification. Available at https://www.tensorflow.org/federated/tutorials/federated_learning_for_image_classification, Last accessed 09.01.2022.

[185] Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Jeremy Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, Mark Purcell, Ambrish Rawat, Tran Ngoc Minh, Naoise Holohan, Supriyo Chakraborty, Shalisha Witherspoon, Dean Steuer, Laura Wynter, Hifaz Hassan, Sean Laguna, Mikhail Yurochkin, Mayank Agarwal, Ebube Chuba, and Annie Abay. IBM federated learning: an enterprise framework white paper V0.1. *CoRR*, abs/2007.10987, 2020.

[186] Holger R. Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, Zhihong Zhang, Wenqi Li, Andriy Myronenko, Dong Yang, Sean Yang, Nicola Rieke, Abood Quraini, Chester Chen, Daguang Xu, Nic Ma, Prerna Dogra, Mona Flores, and Andrew Feng. Nvidia flare: Federated learning from simulation to real-world, 2022.

[187] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *Advances in Neural Information Processing Systems, Best Paper Award at Federate Learning Workshop*, 2020.

[188] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. Fate: An industrial grade platform for collaborative learning with data protection. *J. Mach. Learn. Res.*, 22(1), jul 2022.

[189] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, Théo Ryffel, Zarreen Naowal Reza, and Georgios Kaissis. *PySyft: A Library for Easy Federated Learning*, pages 111–139. Springer International Publishing, Cham, 2021.

[190] Patrick Foley, Micah J Sheller, Brandon Edwards, Sarthak Pati, Walter Riviera, Mansi Sharma, Prakash Narayana Moorthy, Shi-han Wang, Jason Martin, Parsa Mirhaji, Prashant Shah, and Spyridon Bakas. Openfl: the open federated learning library. *Physics in Medicine & Biology*, 2022.

[191] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings, 2018.

[192] Tensorflow. Datasets for running tensorflow federated simulations. Available at https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets, Last accessed 02.21.2022.

[193] Jiahuan Luo, Xueyang Wu, Yun Luo, Anbu Huang, Yunfeng Huang, Yang Liu, and

Qiang Yang. Real-world image datasets for federated learning, 2019.

[194] Rajesh Kumar, Abdullah Aman Khan, Jay Kumar, Noorbakhsh Amiri Golilarz, Simin Zhang, Yang Ting, Chengyu Zheng, Wenyong Wang, et al. Blockchain-federated-learning and deep learning models for covid-19 detection using ct imaging. *IEEE Sensors Journal*, 21(14):16301–16314, 2021.

[195] Fanxing Liu, Cheng Zeng, Le Zhang, Yingjie Zhou, Qing Mu, Yanru Zhang, Ling Zhang, and Ce Zhu. Fedtadbench: Federated time-series anomaly detection benchmark, 2022.

[196] Nikolaos Pitropakis, Emmanouil Panaousis, Thanassis Giannetsos, Eleftherios Anastasiadis, and George Loukas. A taxonomy and survey of attacks against machine learning. *Computer Science Review*, 34:100199, 2019.

[197] Mingfu Xue, Chengxiang Yuan, Heyi Wu, Yushu Zhang, and Weiqiang Liu. Machine learning security: Threats, countermeasures, and evaluations. *IEEE Access*, 8:74720–74742, 2020.

[198] Salem Alqahtani and Murat Demirbas. Performance analysis and comparison of distributed machine learning systems. *arXiv preprint arXiv:1909.02061*, 2019.

[199] Aurick Qiao, Bryon Aragam, Bingjing Zhang, and Eric Xing. Fault tolerance in iterative-convergent machine learning. In *International Conference on Machine Learning*, pages 5220–5230. PMLR, 2019.

[200] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019.

[201] Guan Wang. Interpret federated learning with shapley values. *arXiv preprint arXiv:1905.04519*, 2019.

[202] Fanglan Zheng, Kun Li, Jiang Tian, Xiaojia Xiang, et al. A vertical federated learning method for interpretable scorecard and its application in credit scoring. *arXiv preprint arXiv:2009.06218*, 2020.

[203] Xiaolin Chen, Shuai Zhou, Bei Guan, Kai Yang, Hao Fao, Hu Wang, and Yongji Wang. Fed-eini: An efficient and interpretable inference framework for decision tree ensembles in vertical federated learning. In *2021 IEEE international conference on big data (big data)*, pages 1242–1248. IEEE, 2021.

[204] Zhe Li, Honglong Chen, Zhichen Ni, and Huajie Shao. Balancing privacy protection and interpretability in federated learning. *arXiv preprint arXiv:2302.08044*, 2023.

[205] Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Carl Yang, Han Xie, Lichao Sun, Lifang He, Liangwei Yang, Philip S Yu, Yu Rong, et al. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145*, 2021.

[206] Chaoyang He, Emir Ceyani, Keshav Balasubramanian, Murali Annavaram, and Salman Avestimehr. Spreadgnn: Serverless multi-task federated learning for graph neural networks. *arXiv preprint arXiv:2106.02743*, 2021.

[207] Binghui Wang, Ang Li, Meng Pang, Hai Li, and Yiran Chen. Graphfl: A federated learning framework for semi-supervised node classification on graphs. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 498–507. IEEE, 2022.

[208] Rui Liu and Han Yu. Federated graph neural networks: Overview, techniques and challenges. *arXiv preprint arXiv:2202.07256*, 2022.