DOI:
10.1109/COMST.2023.3330910

Document Version
Peer reviewed version

Federated Learning and Meta Learning: Approaches, Applications, and Directions

Xiaonan Liu, *Member, IEEE,* Yansha Deng, *Senior Member, IEEE,*
Arumugam Nallanathan and Mehdi Bennis, *Fellow, IEEE*

***Abstract*—Over the past few years, significant advancements have been made in the field of machine learning (ML) to address resource management, interference management, autonomy, and decision-making in wireless networks. Traditional ML approaches rely on centralized methods, where data is collected at a central server for training. However, this approach poses a challenge in terms of preserving the data privacy of devices. To address this issue, federated learning (FL) has emerged as an effective solution that allows edge devices to collaboratively train ML models without compromising data privacy. In FL, local datasets are not shared, and the focus is on learning a global model for a specific task involving all devices. However, FL has limitations when it comes to adapting the model to devices with different data distributions. In such cases, meta learning is considered, as it enables the adaptation of learning models to different data distributions using only a few data samples. In this tutorial, we present a comprehensive review of FL, meta learning, and federated meta learning (FedMeta). Unlike other tutorial papers, our objective is to explore how FL, meta learning, and FedMeta methodologies can be designed, optimized, and evolved, and their applications over wireless networks. We also analyze the relationships among these learning algorithms and examine their advantages and disadvantages in real-world applications.**

***Index Terms*—Centralized learning, distributed learning, federated learning, meta learning, federated meta learning, wireless networks.**

I. Introduction

The rapid advancement of technology and the increasing proliferation of devices, including IoT sensors, mobile phones, and tablets, have resulted in an unprecedented growth in data generation [1], [2]. According to a report issued by the SG Analytics, 2.5 quintillion bytes of data were generated every day in 2020 [3]. To extract meaningful information and enable dynamic decision-making for various tasks, machine learning (ML) algorithms are used to analyze these datasets [4], [5], such as controlling self-driving cars [6], pattern recognition [7], or prediction of user behavior [8].

According to [9], in traditional cloud computing systems, various types of datasets, such as images, videos, audio files, and location information, acquired from the internet of thing (IoT) devices or mobile devices, are transmitted to a cloud-based server or data center [10], where centralized ML models are trained. As shown in Fig. 1, in centralized learning, the devices communicate with a central server to upload their data by wired/wireless connections. In particular, the devices transmit their local data to a central server, and the central server performs all the computational tasks to train the data. Then, the output of the centralized learning model is delivered to devices. Centralized learning is computationally-efficient for devices as they are not required to be equipped with high computation capability. However, the traditional centralized training method may not be suitable for the ever-growing complexity and heterogeneity of wireless networks, such as device to device (D2D) and vehicle to vehicle (V2V) communications, as shown in Fig. 1 (b) and (c), respectively, due to the following reasons:

- The presence of limited bandwidth, dynamic wireless channels, and high interference can result in significant transmission latency, which can negatively affect real-time applications, such as wireless virtual reality (VR) [11]–[13], D2D, and self-driving car systems [14]. These applications require immediate and responsive data processing, but the delays caused by limited bandwidth and the unstable wireless channels negatively affect their

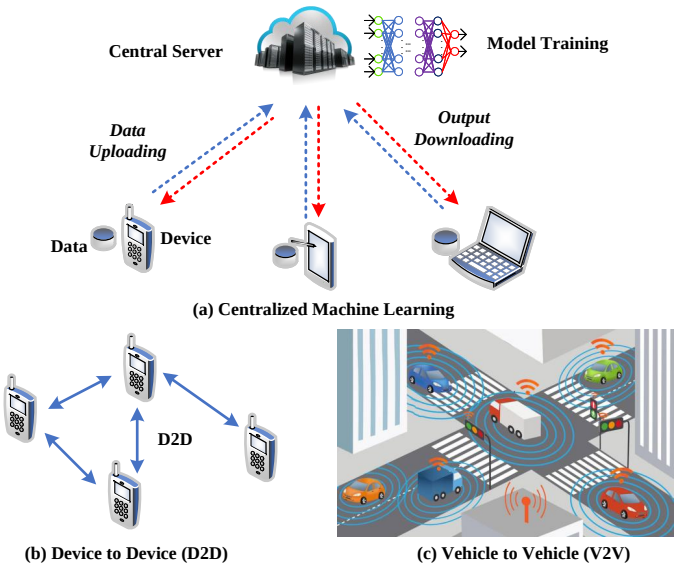


Fig. 1. The canonical architecture of centralized machine learning, device to device communication, and vehicle to vehicle communication.

X. Liu is with the Institute for Digital Communications, The University of Edinburgh, U.K. (e-mail:{xliu8}@ed.ac.uk). (This work is performed at KCL.)

Y. Deng is with the Department of Engineering, King's College London, London, WC2R 2LS, U.K. (e-mail:{yansha.deng}@kcl.ac.uk). (Corresponding author: Yansha Deng).

A. Nallanathan is with the School of Electronic Engineering and Computer Science, Queen Mary University of London (QMUL), U.K. (e-mail:{a.nallanathan}@qmul.ac.uk).

M. Bennis is with Faculty of Information Technology and Electrical Engineering, Centre for Wireless communications, University of Oulu, Finland. (e-mail:{mehdi.bennis}@oulu.fi).

performance and reliability.
- The process of transmitting a large amount of data to the cloud leads to significant communication overhead, as well as increased storage and computational costs [15], [16].
- The data collected from users, such as medical and financial information, can often be private and sensitive in nature. Transmitting this data to the cloud raises concerns on privacy and security, as it exposes users' personal information to potential risks. This situation becomes particularly problematic when it comes to its compliant with data privacy legislation, such as the European Commission's General Data Protection Regulation (GDPR) [17] and the Consumer Privacy Bill of Rights in the U.S. [18].

In order to overcome the aforementioned challenges, distributed learning has emerged as a solution to effectively and efficiently learn models from distributed data. Distributed learning refers to the use of multi-node machine learning algorithms and systems that are specifically designed to address the computational challenges associated with complex centralized algorithms operating on large-scale datasets [19], [20]. By employing distributed learning algorithms, multiple learning models can be trained based on distributed datasets. This approach offers several advantages over centralized approaches, particularly when the number of datasets is large. One notable advantage is the potential for reducing biases, as the distributed learning enables the utilization of diverse datasets and mitigates the impact of individual dataset characteristics [21]–[23].

Conventional machine learning algorithms, such as k-nearest neighbor [24], support vector machine [25], [26], Bayesian networks [27], [28], and decision trees [29], [30], can be trained in a distributed manner by exchanging raw data, which can hardly protect the user data privacy [31]. In order to ensure the privacy of data and facilitate collaborative machine learning (ML) involving complex models distributed across IoT or mobile devices, a method called federated learning (FL) was initially introduced in [32] by McMahan et al. The standard steps of FL include: 1) each device trains a local model using its own dataset; 2) the devices send their local models to a central server for model aggregation; 3) the server updates the global model and transmits it back to the devices. These steps are repeated iteratively until convergence. In addition, FL offers several advantages over traditional centralized learning and distributed learning approaches that rely on the exchange of raw data. Some of these advantages include:

- **Data Privacy:** FL prioritizes user privacy by exchanging only model weights between the server and the devices, rather than the datasets themselves. This approach effectively protects user privacy during the collaborative learning process.
- **Data Diversity:** FL enables the utilization of heterogeneous data by allowing the aggregation of models from different devices, leading to the development of an enhanced global model. This aggregation process enhances the overall performance and effectiveness of the model from various devices.

In addition to the aforementioned advantages, FL has demonstrated its effectiveness in various applications, including training predictive models for human trajectory/behavior by mobile devices [33], automatically learning users' behavior patterns through smart home IoT [34], and enabling collaborative diagnosis in health artificial intelligence (AI) among multiple hospitals and government agencies [35]–[37]. However, these applications usually suffer from data heterogeneity (i.e., data with different statistical characteristics) and system heterogeneity (i.e., systems with different types of computation units), which severely affect the convergence and accuracy of FL algorithms. It is worth noting that FL primarily focuses on learning a single ML task across multiple devices [38], [39]. To address this limitation, integrating meta learning, also known as learning to learn, with FL becomes crucial, resulting in what is known as federated meta learning (FedMeta). FedMeta aims to solve multi-task learning problems. Meta learning aims to create AI models that is able to adapt to new tasks and improve their learning performance over time, without the need for extensive retraining. In other words, meta learning typically involves training a learning model on a variety of different tasks, with the goal of learning generalizable knowledge that can be transferred to new tasks. This is different from traditional ML, where a learning model is typically trained on a single task and then used for that task alone. In general, a meta learning algorithm is trained using the outputs, e.g., model predictions, and metadata of ML algorithms. After finishing training, its learning models are delivered for testing and used to make final predictions. In addition, meta learning includes tasks such as observing the learning performance of different ML models on learning tasks, learning from metadata, and faster learning process for new tasks. While both federated learning (FL) and meta learning involve sharing a global model among multiple devices, they differ in three key aspects: (1) *Data Distribution:* In FL, devices have their own datasets with different distributions but perform the same task. On the other hand, meta learning involves multiple tasks, each with its corresponding dataset. (2) *Update Mechanism:* FL utilizes local updates deployed by devices to enhance learning performance, whereas meta learning employs inner-loop updates for each task to improve learning performance. (3) *Aggregation and Parameter Updates:* FL applies model aggregation to improve the global performance of all devices. In contrast, meta learning employs an outer-loop to update global parameters for all tasks. Initialization-based meta-learning algorithms, in particular, are known for fast adaptation and good generalization to new tasks. These distinctions highlight the different approaches and objectives of FL and meta learning. While FL focuses on collaborative learning with devices having different data distributions but performing the same task, meta learning guarantees the adaptation of models to multiple tasks with their corresponding datasets, often emphasizing fast adaptation and generalization capabilities, especially by initialization-based algorithms [40]. The goal of FedMeta is to collaboratively meta-train a learning model using data from different tasks

distributed among devices. The server maintains the initialized model and updates it by collecting testing loss from a mini batch of devices. The transmitted information in learning process consists of the model parameter initialization (from a server to devices) and testing loss (from devices to the server), and no data is required to be delivered to the server. Compared to FL, FedMeta has the following advantages:

- FedMeta brings a reduction in the required communication cost because of faster convergence, and an increase in learning accuracy. Additionally, it is highly adaptable and can be employed for arbitrary tasks across multiple devices. This flexibility allows for efficient and accurate learning in various scenarios.
- FedMeta enables model sharing and local model training without substantial expansion in model size. As a result, it does not consume a large amount of memory, and the resulting global model can be personalized for each device. This aspect allows for efficient utilization of resources and customization of the global model to satisfy the specific requirements of devices.

A. Related Works

This section provides a brief review of relevant surveys and tutorials on FL and meta-learning. Additionally, it highlights the novel contributions of this paper.

1) FL: In the past 5 years, numerous surveys and tutorials have been published on FL methodologies [41]–[52] and their applications over wireless [53]–[65] networks. To differentiate our tutorial from these existing surveys and tutorials, we have classified them into different categories based on their primary focus in Table I. We then compare and summarize the content of these surveys and tutorials, aligning them with the structure of our own tutorial, as presented in Tables II and III. Table I highlights that surveys and tutorials for FL methodologies primarily focus on either fundamental definitions, architectures, challenges, future directions, and applications of FL [41], [43], [44], or a specific subfield of FL, such as emerging trends of FL (FL in the intersection with other learning paradigms) [45], communication efficiency of FL (challenges and constraints caused by limited bandwidth and computation ability) [46], fairness-aware FL (client selection, optimization, contribution evaluation, incentive distribution, and performance metrics) [47], federated reinforcement learning (FRL) (definitions, evolution, and advantages of horizontal FRL and vertical FRL) [48], FL for natural language processing (NLP) (algorithm challenges, system challenges as well as privacy issues) [49], incentive schemes for FL (stackelberg game, auction, contract theory, Shapley value, reinforcement learning, and blockchain) [50], [51], unlabeled data mining in FL (potential research directions, application scenarios, and challenges) [42], and FL over next-generation Ethernet Passive Optical Networks [52]. On the other hand, FL surveys and tutorials for wireless networks mainly focus on either fundamental theories, key techniques, challenges, future directions, and applications [54]–[57], [64], or a specific subfield of FL applied in wireless networks, including FL in IoT (data sharing, offloading and caching, attack detection, localization, mobile crowdsensing, and privacy) [58], [61], [62], communication-efficient FL under various challenges incurred by communication, computing, energy, and data privacy issues [53], [63], FL for mobile edge computing (MEC) (communication cost, resource allocation, data privacy, data security, and implementation) [59], collaborative FL (definitions, advantages, drawbacks, usage conditions, and performance metrics) [60], and asynchronous FL (device heterogeneity, data heterogeneity, privacy and security, and applications) [65]. To the best of the authors' knowledge, this is the first paper considering FL methodologies in different research areas, including model aggregation, gradient descent, communication efficiency, fairness, Bayesian learning, and clustering, and how FL algorithms evolve from the canonical one, namely, federated averaging (FedAvg), in detail. Also, we present a qualitative comparison considering advantages and disadvantages among different FL algorithms in the same research area.

2) Meta Learning: There have been several surveys and tutorials focusing on meta learning methodologies over the past 20 years [66]–[72]. However, to the best of the authors' knowledge, there are no existing surveys and tutorials in meta learning over wireless networks. To distinguish the scope of our tutorial from existing literature, we classify the available meta learning surveys and tutorials into different categories based on their focus, as presented in Table I. Furthermore, we compare and summarize the content of these existing surveys and tutorials in Table IV, aligning them with the structure and content of our own tutorial. Table I illutrates that the existing meta learning surveys and tutorials focused either on the general advancement, including definitions, models, challenges, research directions, and applications, of meta learning [66], [67], [70], [71], or exploring the detailed applications of meta learning in a specific meta learning field, including algorithm selection (transfer learning, few-shot learning, and beyond supervised learning) for data mining [68], NLP (especially few-shot applications, including definitions, research directions, and some common datasets) [69], and multi-modal meta learning in terms of the methodologies (few-shot learning and zero-shot learning) and applications [72]. To the best of the authors' knowledge, o existing surveys or tutorials have covered the application of FedMeta in wireless networks, making our tutorial a unique contribution in this area of research.

Tables II, III, and IV provide a comparative analysis of existing surveys and tutorials in relation to our tutorial. It is observed that the existing literature only covers a limited number of subtopics related to our tutorial and offers only brief descriptions of the corresponding learning algorithms. In contrast, our tutorial goes beyond these limitations by providing a detailed introduction to the underlying design concepts of the relevant algorithms. Notably, our tutorial stands out by analyzing the relationship and evolution of these algorithms specifically and their applications over wireless networks. This crucial aspect has not been extensively investigated in other surveys or tutorials. By exploring the interplay and advancements of these algorithms and their applications over wireless networks, our tutorial contributes novel insights and addresses research gaps in the existing literature.

TABLE I
AN OVERVIEW OF SELECTED SURVEYS AND TUTORIALS ON FL AND META LEARNING

Subject	Ref.	Contributions
FL Methodologies	[41], [43], [44]	Definitions, architectures, challenges, future directions, and applications of the FL framework
	[45]	Survey on emerging trends in FL, including FL in the intersection with other learning paradigms
	[46]	Survey on communication efficiency of FL, including challenges and constraints caused by limited bandwidth and computation ability
	[47]	Survey on the fairness-aware FL, including client selection, optimization, contribution evaluation, incentive distribution, and performance metrics
	[48]	Survey on federated reinforcement learning (FRL), including definitions, evolution, and advantages of horizontal FRL and vertical FRL
	[49]	Survey on FL for natural language processing (NLP), including algorithm challenges, system challenges as well as privacy issues
	[50], [51]	Surveys on incentive schemes for FL in terms of Stackelberg game, auction, contract theory, Shapley value, reinforcement learning, and blockchain
	[42]	Survey on unlabeled data in FL, including potential research directions, application scenarios, and challenges
	[52]	FL over ethernet passive optical networks, considering dynamic wavelength and bandwidth allocation for quality of service provisioning
FL in Wireless Networks	[54]–[57], [64]	Fundamental theories, key techniques, challenges, future directions, and applications for FL over wireless networks
	[58], [61], [62]	Surveys on FL in IoT, including data sharing, offloading and caching, attack detection, localization, mobile crowdsensing, and privacy
	[53], [63]	Surveys on communication-efficient FL under various challenges incurred by communication, computing, energy, and data privacy issues
	[59]	Survey on FL in MEC, including communication cost, resource allocation, data privacy, data security, and implementation
	[60]	Survey on collaborative FL, including the definitions, advantages, drawbacks, usage conditions, and performance metrics
	[65]	Survey on asynchronous FL, including device heterogeneity, data heterogeneity, privacy and security, and applications
Meta Learning Methodologies	[66], [67], [70], [71]	Introductory tutorial on meta learning, e.g., definitions, models, challenges, research directions, and applications
	[68]	Tutorial on meta learning on algorithm selection (transfer learning, few-shot learning, and beyond supervised learning) for data mining
	[69]	Survey on meta learning for NLP, especially few-shot applications, including definitions, research directions, and some common datasets
	[72]	Tutorial on multimodality-based meta-learning in terms of the methodologies (few-shot learning and zero-shot learning) and applications

TABLE II
COMPARISON OF SURVEYS AND TUTORIALS ON FL METHODOLOGIES

Reference		[41]	[42]	[43]	[44]	[45]	[46]	[47]	[48]	[49]	[50]	[51]	This paper
Year		2019	2020	2021	2021	2021	2021	2021	2021	2021	2021	2021	-
Section III: FL Methodologies	(III.A): Model Aggregation	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓
	(III.B): Gradient Descent	✓	✓	✓	✓				✓	✓	✓		✓
	(III.C): Communication Efficiency						✓						✓
	(III.D): Fairness					✓		✓			✓		✓
	(III.E): Bayesian Machinery					✓					✓		✓
	(III.F): Clustering					✓							✓

TABLE III
COMPARISON OF SURVEYS AND TUTORIALS ON FL OVER WIRELESS NETWORKS

Reference		[53]	[54]	[55]	[56]	[57]	[58]	[59]	[60]	[61]	[62]	[63]	[64]	[65]	This paper
Year		2019	2020	2020	2020	2020	2020	2020	2020	2021	2021	2021	2021	2021	-
Section IV: FL in Wireless Networks	(IV.A.1): Device Selection					✓	✓	✓		✓			✓		✓
	(IV.A.2.a): Communication Efficiency	✓		✓		✓		✓		✓	✓	✓	✓		✓
	(IV.A.2.b): Computation Efficiency			✓				✓				✓			✓
	(IV.A.2.c): Energy Efficiency									✓		✓			✓
	(IV.A.2): Packet Error					✓				✓			✓		✓
	(IV.A.3): Resource Allocation		✓	✓	✓			✓		✓	✓		✓		✓
	(IV.A.4): Asynchronous					✓		✓	✓	✓				✓	✓
	(IV.B): Over the Air Computation				✓		✓					✓	✓		✓

B. Summary of Contributions

Given the privacy-preserving characteristics of federated learning (FL) and the ability of meta learning to quickly adapt to different tasks, researchers from both academia and industry are now exploring the joint design of FL, meta learning, and federated meta learning (FedMeta) and wireless networks. This paper provides the first comprehensive tutorial that highlights the research areas, relationships, advancements, challenges, and opportunities associated with these three learning concepts and their applications over wireless network environments. The main contributions of this article can be summarized as follows:

- Based on the foundational FedAvg algorithm, we outline six key research areas of FL methodologies, including model aggregation, gradient descent, communication efficiency, fairness, Bayesian learning, and clustering. We provide a detailed analysis of the relationships and evolutionary developments of the respective learning algorithms within these areas. Furthermore, we introduce two research areas that explore the interplay between FL and wireless factors over wireless networks, including digital and analog over-the-air computation schemes.
- We summarize three key research areas in meta-learning, including metric-based, model-based, and gradient-based

TABLE IV
COMPARISON OF SURVEYS AND TUTORIALS ON META LEARNING

Reference		[66]	[67]	[68]	[69]	[70]	[71]	[72]	This paper
Year		2002	2015	2018	2020	2020	2020	2021	-
Section V: Meta Learning Methodologies	(V.A.1): Siamese			✓	✓	✓	✓	✓	✓
	(V.A.2): Matching			✓	✓	✓		✓	✓
	(V.A.3): Prototypical			✓	✓	✓	✓	✓	✓
	(V.A.4): Relation				✓	✓	✓	✓	✓
	(V.B.1): Memory-augmented Neural Network			✓			✓		✓
	(V.B.2): Meta Network								✓
	(V.B.3): Recurrent Meta-learner			✓					✓
	(V.B.4): Simple Neural Attentive Meta-learner			✓			✓		✓
	(V.C.1): Model-Agnostic Meta-Learning (MAML)			✓	✓	✓	✓	✓	✓
	(V.C.2): Meta-SGD					✓	✓		✓
	(V.C.3): Reptile			✓			✓	✓	✓
	(V.C.4): Bayesian MAML					✓	✓		✓
	(V.C.5): Laplace Approximation for Meta Adaptation						✓		✓
	(V.C.6): Latent Embedding Optimization						✓		✓
	(V.C.7): MAML with Implicit Gradients								✓
	(V.C.8): Online MAML					✓	✓		✓
Section VI: Meta Learning in Wireless Networks	(VI.A): Traffic Prediction								✓
	(VI.B): Rate Maximization								✓
	(VI.C): MIMO Detectors								✓

meta-learning. Based on the gradient-based meta-learning paradigm, we focus on the fundamental scheme known as model-agnostic meta-learning (MAML). We discuss the evolution of MAML, providing a detailed analysis of its advancements and applications. Additionally, we explore the potential of meta-learning in solving wireless communication problems. By leveraging meta-learning techniques, we investigate how they can be utilized to tackle various challenges and optimize wireless communication systems.

- The fundamental principle of federated meta learning (FedMeta) and its evolution are comprehensively summarized. Furthermore, we introduce two specific research areas of FedMeta over wireless networks, including device selection and energy efficiency. We explore how FedMeta can be applied to address challenges related to device selection, such as optimizing the selection of devices for participation in the FL process. Additionally, we discuss how FedMeta can contribute to enhancing energy efficiency in wireless networks, thereby improving the overall energy consumption of the system.
- In addition to the previously discussed topics, we present several other important aspects related to the implementation of FL, meta-learning, and FedMeta. We explore different implementation platforms that facilitate the practical deployment of these learning methodologies. Furthermore, we present real-world applications where FL, meta-learning, and FedMeta have demonstrated their effectiveness and potential impact. In addition, we identify and highlight open research problems that present exciting opportunities for future research in the field of FL, meta-learning, and FedMeta. These research problems have the potential to drive innovation and pave the way for new directions and advancements in these learning concepts. By addressing these open challenges, researchers can contribute to the further development and practical applications of FL, meta-learning, and FedMeta.

The rest of this paper is organized as follows. In Section II, we introduce the background and fundamentals of distributed learning. In Sections III and IV, we present important research fields in FL methodologies and their applications over wireless networks, respectively. In Sections V and VI, we introduce research areas in meta learning methodologies and their applications over wireless networks, respectively. Section VII presents principle of FedMeta and its applications over wireless networks. Section VIII introduces open research problems and future directions in FL, meta learning, and FedMeta. A graphical illustration of the content of Sections II to VII is provided in Fig. 2. Finally, conclusions are drawn in Section IX.

II. BACKGROUND AND FUNDAMENTALS OF DISTRIBUTED LEARNING

In distributed learning, two main approaches are commonly used for learning across servers or devices: data parallelizing and model parallelizing. In the data parallelizing approach, the data is divided into multiple datasets, and each server or device applies the same machine learning (ML) algorithm to a different dataset. This approach allows for parallel processing of data, enabling faster training and improved scalability. On the other hand, the model parallelizing approach involves segmenting the ML model into different sub-models. Each sub-model is updated on different servers or devices, and therefore global model is therefore the aggregation of all sub-models. However, it is important to note that not all ML algorithms are compatible with the model parallel approach due to the specific requirements and dependencies of the model. It is also worth mentioning that data parallelizing and model parallelizing can be employed simultaneously, where both the data and the model are distributed across multiple servers or devices. This hybrid approach leverages the advantages of both techniques to achieve improved performance and efficiency in distributed learning scenarios [73]. In this section, we will provide a comprehensive description of the background and

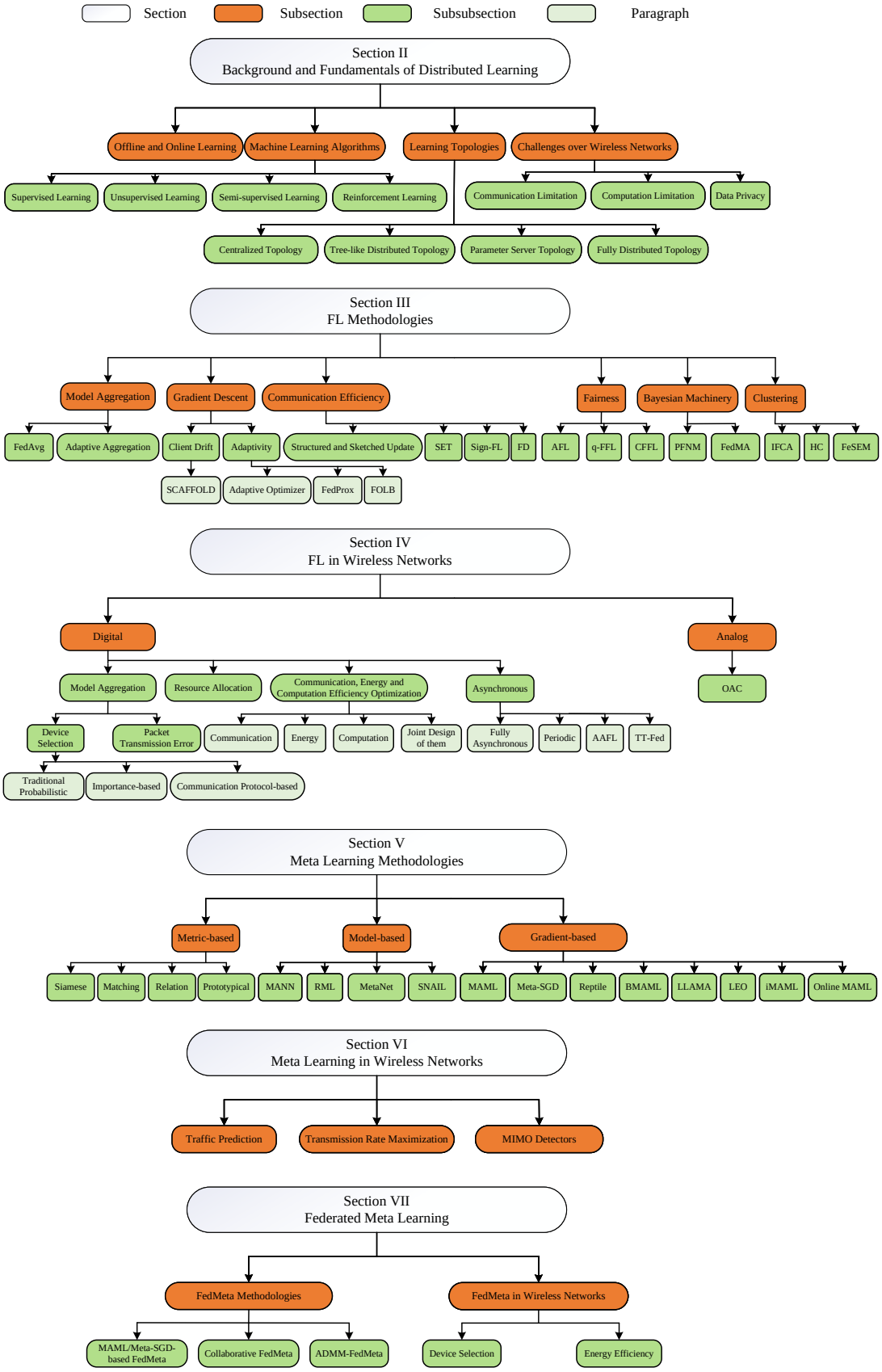


Fig. 2. Organization and content of Sections II to VII.

fundamentals of distributed learning, including offline and online learning, learning topologies, challenges of distributed learning in wired/wireless networks, and application of FL in ML paradigms.

A. Offline and Online Learning

Based on whether the learning model is updated with newly arriving data [74], distributed learning algorithms can be categorized into offline learning and online learning. Offline learning refers to the process of updating the learning model using the knowledge derived from previous observations. The objective of offline learning is to maximize accuracy or long-term reward for prediction or decision-making tasks, leveraging the information gathered from historical data. On the other hand, online learning involves continuously updating the learning model in response to newly arriving data and observations from the environment. The primary goal of online learning is to optimize prediction or decision-making performance in real-time, adapting the model to changing circumstances. Both offline and online learning methods are designed to improve the learning accuracy and long-term reward of prediction or decision-making tasks. While offline learning focuses on leveraging historical observations, online learning emphasizes adaptability to real-time data streams and the ability to quickly update the model to capture changing patterns and dynamics in the environment.

Offline learning consists of two phases, namely, the training phase and the testing phase. In the training phase, the learning model is first trained with the training dataset until an optimal set of hyperparameters achieving the highest accuracy or reward for the given task. In the testing phase, the trained learning model is employed for prediction or decision-making without any further updates. The model utilizes its learned knowledge to make predictions or decisions on new data. The offline training method suffers from high re-training costs when dealing with new training data/environments, and thus, has poor scalability for real-world applications, especially when the amount of data grows and the environment evolves rapidly [75].

Online learning can be applied to a wide range of ML algorithms, where the learning models are trained to handle prediction or decision-making tasks by continuously learning from a sequence of data samples in a sequence manner, one by one, in each time slot. Online learning addresses the limitations of offline learning by enabling the learning models to be updated constantly and efficiently as new training data becomes available. This constant updating of the learning models in online learning makes them highly efficient and scalable for large-scale ML tasks in real-world applications. By incorporating new training data in a timely manner, online learning models can adapt to evolving environments and changing data distributions, enhancing their ability to handle real-time and dynamic scenarios [76].

To efficiently solve prediction/decision-making problems, the servers and devices need to be connected and communicate with each other for information exchange, which can be represented via the topology of the distributed ML systems. In the following subsections, various types of topologies are introduced in detail.

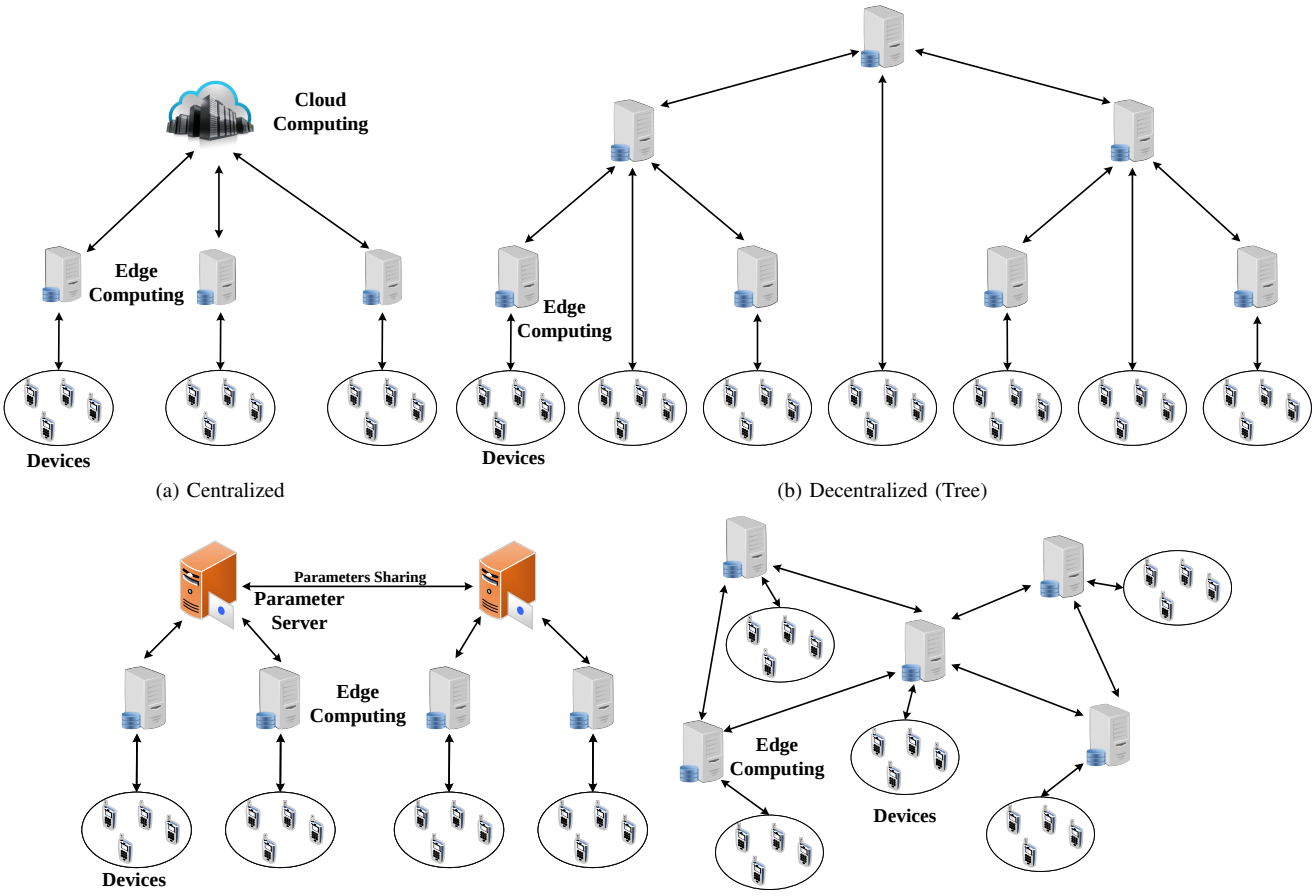
B. Learning Topologies

The selection of topology plays an important role in designing distributed ML systems. The topology determines how servers and devices are organized and connected within the system. Various factors, including the degree of distribution in transmitting learning models, influence the selection of a suitable topology. In this subsection, four general learning topologies in distributed ML are introduced and shown in Fig. 3, as proposed in [77] and [78].

1) Centralized Topology: The centralized topology is characterized by a central server that is connected to multiple edge servers to obtain learning models, as shown in Fig. 3 (a). The central server first obtains learning models from all edge servers, and the the model aggregation is executed at the central server. Next, the central server transmits the aggregated learning model to all edge servers. Each edge server then broadcasts the learning model to the devices it is connected to, and each device updates the transmitted learning model based on its local dataset.

2) Tree-like Distributed Topology: The tree-like distributed topology is characterized by a hierarchical arrangement of connected nodes, resembling the branches of a tree, as shown in Fig. 3 (b). There is only one connection between any two connected nodes, establishing a natural parent and child hierarchy. Therefore, parameters are shared between parent and child nodes. The tree-like distributed topology offers advantages in scalability and manageability. Communication is simplified since each edge server or device only needs to interact with its parent and child servers, reducing the overall complexity of information exchange and coordination [79]. In this topology, the edge servers within the tree structure accumulate the local gradients computed by their child servers. These accumulated gradients are then passed up to their parent servers to calculate a global gradient. This process enables collaborative gradient computation across the tree-like structure, allowing the learning models to collectively update and optimize based on the information exchanged between parent and child nodes.

3) Parameter Server Topology: The parameter server topology consists of a distributed set of edge servers and a centralized set of parameter servers responsible for maintaining and sharing the models [80], [81], as shown in Fig. 3 (c). All learning models are stored in the parameter servers using a global shared memory. Edge servers have direct access to the models stored at the parameter servers and can obtain and update the models as needed in any time slot. Meanwhile, edge devices communicate and share parameters with edge servers. The advantage of this topology lies in the direct accessibility of models for all edge servers. However, one drawback of this topology is the communication overhead. As the parameter servers handle all communication, there will be a large amount of communication overhead between the edge servers and the parameter servers [82].



(a) Centralized

(b) Decentralized (Tree)

(c) Decentralized (Parameter Server)

(d) Fully Distributed

Fig. 3. Centralized and distributed machine learning topologies.

4) Fully Distributed Topology: The fully distributed topology does not have any central servers, as shown in Fig. 3 (d). Instead, it consists of a set of independent edge servers, each responsible for updating its own learning model and directly exchanging its model with other edge servers. One of the key advantages of the fully distributed topology is its high scalability. Additionally, the fully distributed topology exhibits robustness against failures of individual servers, such as power outages or malfunctions [83]. However, it's important to note that the fully distributed topology can incur significant communication overhead, especially when a large number of edge servers are present in the network.

To solve the high communication overhead problem in distributed topologies, a fast, and communication-efficient distributed framework, so-called group alternating direction method of multipliers (GADMM), was proposed in [84]–[86]. In GADMM, at most half of the edge servers are competing for the limited communication resources in each time slot. Meanwhile, each edge server exchanges the trained model only with two neighboring edge servers, thereby training a global model with a lower amount of communication overhead in each exchange.

C. Challenges of Distributed Learning

In wired/wireless networks, it is obvious that the overall communication and computation overhead increases as the number of servers and devices grows. To develop effective distributed ML algorithms for wired/wireless networks, three primary factors should be considered, including the communication cost, computation cost, and data privacy.

1) Wireless Communication Limitation: In wireless networks, a number of servers and devices may share the same spectrum resource due to the limited bandwidth [87]. Therefore, the communication among the devices and edge servers may suffer from high interference, poor channel conditions, and noise, which lead to low reliability, high transmission latency, and low learning accuracy [88].

2) Computation Limitation: Training and operating ML algorithms usually require computation units with high processing capability, especially when the ML models are complex. However, the devices have limited computation and energy capabilities. To minimize the computation latency at the device side, the use of edge servers with high processing capability using graphics processing units (GPUs) have been recently proposed to move the computations from the device to the edge.

3) Data Privacy: Transmitting datasets of edge devices to edge servers can cause data breach if the datasets have

privacy-sensitive information, one potential solution is to only exchange the weights of ML models. Nevertheless, it is possible that the transmitted model parameters can be reversely traced, so that the privacy is still not preserved [89].

In order to address the challenges related to communication, computation, and data privacy, FL has emerged as an effective approach to exploit the distributed devices to collaboratively train ML models. FL was first introduced by Google in 2016, where multiple devices jointly train an ML model without sharing their private data, under the supervision of a central server. This ensures the privacy of the training data of all devices. FL has two entities: a centralized server that owns the global model and a set of devices that store the local models and training datasets. Meanwhile, FL consists of four procedures [90]: 1) training the local model based on the local dataset at the local device; 2) transmitting the local models from the devices to the central server; 3) aggregating the local models to a global model at the central server; and 4) updating the received global model from the central server at devices. The original data of each device is kept locally and does not need to be exchanged or migrated between devices, which ensures the privacy of each device. As a result, devices can benefit from the advantages of shared models trained by other devices without data sharing.

D. *Application of FL in Machine Learning Paradigms*

ML algorithms have the ability to make predictions or decisions by learning from datasets or observed states in the environment. To train a learning model, feedback is required to iteratively improve the learning model. ML algorithms can be classified into different categories based on the type of feedback they receive, including supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning algorithms.

1) Supervised Learning: Supervised learning, one of the fundamental ML approaches, involves training a function that can map inputs to outputs based on labeled data. Each training data sample consists of an input object, typically represented as a vector, along with its corresponding desired output value. The objective of the learning process is to minimize the error between the true data label and the predicted output. By analyzing the training data, the learning model can generalize and accurately predict labels for new datasets. Unfortunately, it is worth noting that as the volume of training data increases, more complex models may be required to achieve accurate predictions [91], [92]. FL is commonly used in tasks where labeled data is readily available, which aligns well with supervised learning tasks. However, considering the challenge of limited labeled data in FL scenarios, the exploration of unsupervised and semi-supervised FL algorithms becomes essential.

2) Unsupervised Learning: Unsupervised learning is an ML approach that involves learning from unlabeled data samples. It utilizes ML algorithms to analyze and cluster unlabeled datasets, which can discover hidden patterns or data groupings without the need for human intervention. One prominent example of an unsupervised learning technique is Principal Component Analysis (PCA) [93]. PCA transforms high-dimensions data into lower-dimensions and it combines the original features into a new feature space by a projection direction that most information in the new feature space is retained from the original data [94]. A common application of unsupervised learning is clustering. In clustering, the learning model automatically groups the training data into groups with similar features. In [95], a federation of unsupervised learning (FedUL) was proposed to verify the possibility of unsupervised FL, where the unlabeled data were transformed into surrogate labeled data for each device, a modified model was trained by supervised FL, and the desired model was recovered from the modified model. FedUL is a very general solution to unsupervised FL. It is compatible with many supervised FL methods, and the recovery of the wanted model can be theoretically guaranteed as if the data have been labeled. In [96], federated PCA in a memory-limited setting was proposed, which provided robustness against stragglers. In [97], federated PCA for vertically partitioned dataset method was considered, which reduced the dimensionality across the joint datasets over all devices and extracted the principal component feature information for downstream data analysis.

3) Semi-supervised Learning: Semi-supervised learning is a learning approach that leverages both labeled and unlabeled data to perform learning tasks, and is conceptually sitting between supervised and unsupervised learning. By utilizing a large amount of unlabeled data in combination with smaller sets of labeled data, semi-supervised learning enables the learning model to accurately classify unlabeled data, leading to a significant improvement in learning accuracy [98]. In [99], semi-supervised FL (SemiFed) was proposed to unify two dominant approaches for semi-supervised learning, which are consistency regularization and pseudo-labeling. SemiFed first performs consistency regularization, which encourages the network to produce similar output distributions when its inputs are perturbed. This regularization can be applied to all samples without labels. Following several rounds of training, the concept of pseudo-labeling is employed. Pseudo-labeling involves assigning artificial labels to unlabeled images and then training the model to predict these artificial labels when fed with unlabeled samples as input in the following training stages.

4) Reinforcement Learning: Reinforcement learning (RL) is an ML approach in which agents make decisions based on the observations obtained from the environment, aiming to maximize the long-term reward [100]. Unlike supervised learning, RL does not rely on labeled input and output data. Instead, it focuses on striking a balance between exploration (of unknown territory) and exploitation (of current knowledge), and training ML models to make a sequence of decisions [101]. However, implementing RL in practical scenarios has several challenges. For example, it is impossible to explore the entire state-action spaces comprehensively. Although distributed RL algorithms can help solve the problem, they usually require data collection from each agent, which may compromise agent privacy and lead to information leakage. To address these privacy concerns, federated reinforcement learning (FRL) has been proposed [48]. FRL not only enables agents to learn

Algorithm 1 Gradient Descent Algorithm

1: Initialize learning weights $\boldsymbol{w}$ and learning rate η.
2: **for** Iteration = 1,...,I **do**
3: **for** i = 1,...,N **do**
4: Input the data x_i.
5: Obtain the loss $l_i(\boldsymbol{w})$.
6: **end for**
7: Calculate the loss function $\mathcal{L}(\boldsymbol{w})$ for the input data based on (1).
8: Update the learning weights $\boldsymbol{w}$ based on (2).
9: **end for**

optimal decisions in unknown environments but also ensures that the privately collected data during an agent's exploration does not need to be shared with others.

In the following two sections, we introduce several main research areas in FL methodologies and their applications over wireless networks. In particular, we first present six FL research areas in FL methodologies. Then, we present two FL research areas in wireless networks.

III. FL Methodologies

In this section, we introduce six main FL research aspects of FL methodologies, including model aggregation, gradient descent, communication efficiency, fairness, Bayesian machinery, and clustering. The transmission between servers and devices is error-free without considering any wireless factors. For ML algorithms, given a labeled set of inputs and their corresponding outputs, the learning models are trained and tested, as shown in Fig. 4. The goal of ML algorithms is to minimize the loss function designed based on a specific learning problem. The loss function plays an important role in training ML algorithms and improvement of their performance [102], and is expressed as

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} l_i(\boldsymbol{w}), \tag{1}$$

where N is the number of data samples, and $l_i(\boldsymbol{w})$ is the loss of the ith input data based on learning weights $\boldsymbol{w}$. To find the minimum value of the loss function, gradient descent is introduced to calculate the derivative of the loss function via $\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}$. Then, the weights of the learning models are updated as

$$\boldsymbol{w} = \boldsymbol{w} - \eta \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}}, \tag{2}$$

where η is the learning rate. After a sufficient number of iterations, the loss function can achieve its minimum value, and the ML model converges. The gradient descent algorithm is presented in **Algorithm 1**.

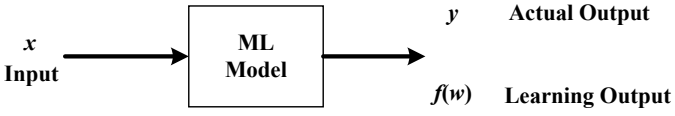


Fig. 4. The architecture of the ML process.

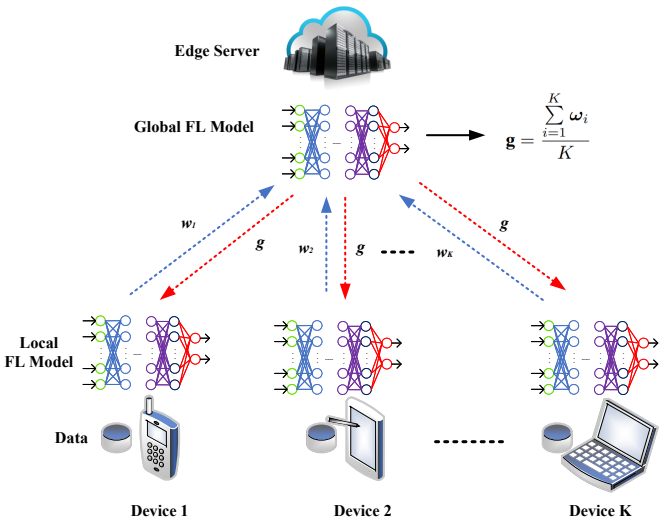


Fig. 5. The canonical architecture of FedAvg.

FL, as first proposed by McMahan in [32], uses the loss function and gradient descent algorithm based on (1) and (2) to collaboratively learn a shared learning model while keeping all the training data at the devices.

We assume that $\hat{K}$ devices have their own datasets $\{D_1, ..., D_{\hat{K}}\}$, and each of them cannot access to the other devices' datasets. As shown in Fig. 5, FedAvg can learn a model by aggregating learning models from distributed devices. In each iteration, $K(K \leq \hat{K})$ devices are selected, and each device first performs the local gradient descent for its own dataset. Then, the local updated model weights $\boldsymbol{w}_k(k = 1, 2, ..., K)$ are transmitted to the server. The server aggregates the local models $\{\boldsymbol{w}_1, ..., \boldsymbol{w}_K\}$ to a global model $\boldsymbol{w}_G$, and transmits the global model to all devices to replace the local model [103].

A. Model Aggregation

Model aggregation is the process of integrating models from multiple devices in order to create a new model [104]–[106]. There are several ways for model aggregation as explained in the following.

Federated Averaging: The most basic model aggregation method is the Federated Averaging (FedAvg) algorithm proposed in [32], where the weights of local models are averaged at the central server to update the global model [107]. We define p_k as the percentage of the number of data samples at the kth device over the total number of data samples at all devices, n_k as the number of data samples of dataset D_k, and n as the total number of data samples, which is calculated as $n = \sum_{k=1}^{K} n_k$. The training objective of FedAvg is given as follows [32, Eq. (1)]

$$\min_{\boldsymbol{w}} \ \mathcal{L}(\boldsymbol{w}_G) = \sum_{k=1}^{K} p_k f_k(\boldsymbol{w}_k), \tag{3}$$

where

$$p_k = \frac{n_k}{n} \quad \text{and} \quad f_k(\boldsymbol{w}_k) = \frac{1}{n_k}\sum_{i \in D_k} l_i(\boldsymbol{w}_k). \tag{4}$$

In (4), $l_i(\boldsymbol{w}_k)$ is the loss of the FL model due to the ith data sample calculated by the local model weights $\boldsymbol{w}_k$ of the kth device, and $\boldsymbol{w}_G$ is the weight of the global model.

There are two approaches to update the global model. The first approach is to compute the gradient of each device. Then, the central server aggregates these gradients from K devices and updates the global model using

$$\boldsymbol{w}_G^{t+1} = \boldsymbol{w}_G^t - \eta \sum_{k=1}^{K} p_k \mathbf{g}_k^t, \quad \text{with } \mathbf{g}_k^t = \nabla f_k(\boldsymbol{w}_k^t), \tag{5}$$

where $\nabla f_k(\boldsymbol{w}_k^t)$ is the gradient computed at the kth device. The second approach is to update the weights of the local model at each device using

$$\boldsymbol{w}_k^{t+1} = \boldsymbol{w}_G^t - \eta \mathbf{g}_k^t, \tag{6}$$

where $\mathbf{g}_k^t$ is the gradient calculated using (5). Then, the global model at the central server is updated as

$$\boldsymbol{w}_G^{t+1} = \sum_{k=1}^{K} p_k \boldsymbol{w}_k^{t+1}. \tag{7}$$

In the second approach, each device first performs the gradient descent for its local model using the local datasets, and the central server averages these local models. In this way, each device can iterate the local update in (6) multiple times before uploading local models, which can accelerate the convergence speed.

Although FedAvg has achieved great success and is one of the most well-known algorithms in FL, the statistical heterogeneity challenges in the data are still difficult to overcome. That is to say, the training data follow a non-independent and non-identical distribution (non-IID), which negatively affects the convergence behavior. To address this issue, adaptive aggregation is introduced.

Adaptive Aggregation: Different from the FedAvg, adaptive aggregation uses a different way to update the global model. To improve the accuracy and convergence performance of the global model, a temporally weighted aggregation method utilizing the previously trained local models was proposed in [108]. The authors in [108] assumed that the local models updated in the $(t-i)$th time slot $(i = 1, ..., t-1)$ are less important than those updated in the tth time slot. In practice, the training data at each device changes over time, and the local models that are more recently updated have a higher importance during model aggregation. In order to account for the freshness of the local models, the global model is updated using [108, Eq. (1)]

$$\boldsymbol{w}_G^{t+1} = \sum_{k=1}^{K} p_k \left(\frac{e}{2}\right)^{-(t-\hat{t}^k)} \boldsymbol{w}_k, \tag{8}$$

where p_k is given in (4), e is the scalar constant used to represent the time effect, and $\hat{t}^k$ is the time slot in which the newest $\boldsymbol{w}_k$ is updated. By introducing parameters $\left(\frac{e}{2}\right)^{-(t-\hat{t}^k)}$, the impact of the local models updated in previous time slots is reduced, and the global model updated in the current time slot is weighted to be more important for the new data, which results in higher learning accuracy.

In [109], an adaptive weighting approach, namely, Inverse Distance Aggregation (IDA), was proposed. The global model updating still follows (5) or (7). Compared with the FedAvg, the main difference of the IDA is the manner in which the weighting coefficient p_k is calculated, which is based on the inverse distance of the local model weights to the global model weights. To realize this, the l_1-norm is used as a metric to measure the distance between the weights of the local model of the kth device $\boldsymbol{w}_k$ and that of the global model $\boldsymbol{w}_G$, and p_k is calculated as

$$p_k = \frac{\|\boldsymbol{w}_G^t - \boldsymbol{w}_k^t\|_1}{\sum_{k=1}^{K} \|\boldsymbol{w}_G^t - \boldsymbol{w}_k^t\|_1}. \tag{9}$$

Calculating p_k via (9) allows us to give higher weight to devices whose distance between the weights of the local model and the weights of the global model are higher. It is important to note that the IDA approach is based on the assumption that devices with more data should have a greater contribution in updating the weights to the local model.

A novel layer-wise adaptive aggregation scheme was proposed in [110] to iteratively update weights while attempting to reduce the distance between the global model and local models. Although the global model is still updated using either (5) or (7), p_k in each layer in [110] is calculated to minimize the distance between each layer of the local model and each layer of the global model. For the lth layer, the weighting coefficient $p_{k,l}$ is calculated as

$$p_{k,l} = \text{softmax}(s_{k,l}^t) = \frac{e^{s_{k,l}^t}}{\sum_{i=1}^{K} e^{s_{i,l}^t}}, \tag{10}$$

where

$$s_{k,l}^t = \|\boldsymbol{w}_{G,l}^t - \boldsymbol{w}_{k,l}^t\|_d. \tag{11}$$

Here, d refers to the l_d-norm, which is used to calculate the distance $s_{k,l}^t$ between the lth layer of the global model and the lth layer of the local model of the kth device. The softmax function in (10) is applied to guarantee $p_{k,l}$ in the range of 0 to 1. This is because the softmax function is a function that converts a vector of $\tilde{N}$ real values into a vector of $\tilde{N}$ real values that sum to 1 [111]. The advantage of the layer-wise adaptive FL is that it can minimize the distance between the global model and the local models. The summary of FL algorithms in model aggregation is summarized in Table V.

B. Gradient Descent

Standard federated optimization methods, such as FedAvg [32], may show unfavorable convergence performance, especially in heterogeneous networks. This is mainly caused by two factors: 1) client drift, where the local models move away from the optimal global model, which can lead to unstable and slow convergence; and 2) lack of adaptivity, where the FedAvg may be unsuitable for datasets with heavy-tailed stochastic gradient noise distributions, this often happens in NLP research [112]. Heavy-tailed distributions are probability distributions whose tails are not exponentially bounded, that is to say, they have

TABLE V
SUMMARY OF FL ALGORITHMS IN MODEL AGGREGATION

Model Aggregation Algorithm	Advantage	Disadvantage	Condition
Federated Averaging [32]	Guarantee device privacy	Poor adaption to system and statistical heterogeneity	Synchronous mode
Temporally Weighted Aggregation [108]	Low communication costs High learning accuracy	Poor adaption to asynchronous mode	Synchronous mode
Inverse Distance Aggregation [109]	Adapt to statistical heterogeneity High learning accuracy	Not robust to low quality and poisonous data	Synchronous mode Devices have enough data
Layer-wise Adaptive Aggregation [110]	Consider the relation between the server and device models Server model is well-generalized Low communication costs	Poor adaption to asynchronous mode Performance on non neural language model is unknown	Execute on neural language model

heavier tails than the exponential distribution [113]. Several novel gradient descent methods have been proposed to solve the client drift and lack of adaptivity problems, which are introduced in the following:

1) Client Drift: To mitigate the problem of client drift, a new Stochastic Controlled Averaging algorithm (SCAFFOLD) was proposed in [114], where control variates for the kth device $\mathbf{c}_k$ and the variate for the server $\mathbf{c}_G = \frac{1}{K}\sum_{i=1}^{K}\mathbf{c}_i$ were used in the gradient descent to update the local and global models, respectively. Unlike the FedAvg, the gradient descent of the kth device of the SCAFFOLD algorithm is given by

$$\boldsymbol{w}_k^{t+1} = \boldsymbol{w}_k^t - \eta(\mathbf{g}_k^t + \mathbf{c}_G^t - \mathbf{c}_k^t), \tag{12}$$

where $\mathbf{c}_G^t - \mathbf{c}_k^t$ guarantees the gradient descent moving towards the right direction, and $\mathbf{c}_k^{t+1}$ is calculated using

$$\mathbf{c}_k^{t+1} = \mathbf{c}_k^t - \mathbf{c}_G^t + \frac{1}{\tilde{N}_k\eta}(\boldsymbol{w}_G^t - \boldsymbol{w}_k^t). \tag{13}$$

In (13), SCAFFOLD uses the previous computed gradients to update the control variate, and $\tilde{N}_k$ is the number of updating iteration of the kth device with its local data in the tth time slot. Then, the global control variate $\mathbf{c}_G$ is aggregated as

$$\mathbf{c}_G^{t+1} = \mathbf{c}_G^t + \frac{1}{K}\sum_{i=1}^{K}(\mathbf{c}_i^{t+1} - \mathbf{c}_i^t). \tag{14}$$

The correction term $(\mathbf{c}_G - \mathbf{c}_k)$ in (12) ensures that the local model updates move towards the optimal direction, so as to address the client drift issue of FedAvg.

2) Adaptivity: An adaptive learning algorithm has adaptive learning parameters, such as learning rate, which can automatically adjust the statistics of the received data, available computational resources, or other information related to the environment in which it operates. Adaptive variants can help to learn algorithms to improve the convergence performance and learning accuracy [115]. To improve the convergence performance of FedAvg, three methods have been proposed, namely, Adaptive Optimizer, Federated Proximal (FedProx), and Fast-convergent FL (FOLB).

a) Adaptive Optimizer: The SGD in FedAvg may be unsuitable for settings with heavy-tailed stochastic gradient noise distributions. To address this issue, traditional adaptive optimization algorithms, such as Adagrad, Adam, and Yogi have been integrated into FL to update the global model in [116]. The global model is updated as

$$\boldsymbol{w}_G^{t+1} = \boldsymbol{w}_G^t + \frac{\eta}{\sqrt{v^t} + \tau}\hat{\boldsymbol{w}}_t, \tag{15}$$

where $v^t = \{v_{\text{Adagrad}}^t, v_{\text{FedAdam}}^t, v_{\text{FedYogi}}^t\}$ are exponential moving averages (EMAs) of the gradients of the FedAdagrad, FedAdam, and FedYogi optimization methods, η is the learning rate, and τ controls the degree of adaptivity of the algorithm, where smaller values of τ denotes higher degrees of adaptivity. EMA gives a higher weight to the most recent data points. In (15), $\hat{\boldsymbol{w}}_t$ is the sum of the past gradient differences between the local and global model for given decay parameters $\beta_1 \in [0, 1)$ and $\beta_2 \in [0, 1)$, and is calculated as

$$\hat{\boldsymbol{w}}_t = \beta_1\hat{\boldsymbol{w}}_{t-1} + (1-\beta_1)\left(\frac{1}{K}\sum_{i=1}^{K}\triangle\boldsymbol{w}_i^t\right). \tag{16}$$

In (16), the difference $\triangle\boldsymbol{w}_i^t$ between the local and global model of the ith device in the tth time slot is calculated as

$$\triangle\boldsymbol{w}_i^t = \boldsymbol{w}_i^t - \boldsymbol{w}_G^t. \tag{17}$$

FedAdam was proposed to address the problem of the rapid decay of the learning rate of the FedAdagrad optimization algorithm. However, FedAdam leads to a situation where the past gradients are forgotten in a fairly fast manner, which can be especially problematic in sparse settings, where gradients are rarely non-zero [117]. To solve the problem caused by FedAdam, a simple adaptive method called FedYogi was proposed.

For the FedAdagrad optimizer, v_{Adagrad}^t is given by

$$v_{\text{Adagrad}}^t = v_{\text{Adagrad}}^{t-1} + \|\hat{\boldsymbol{w}}_t\|^2. \tag{18}$$

For the FedAdam optimizer, v_{FedAdam}^t is written as

$$v_{\text{FedAdam}}^t = \beta_2 v_{\text{FedAdam}}^{t-1} + (1-\beta_2)\|\hat{\boldsymbol{w}}\|_t^2. \tag{19}$$

For the FedYogi optimizer, v_{FedYogi}^t is obtained as

$$v_{\text{FedYogi}}^t = v_{\text{FedYogi}}^{t-1} - (1-\beta_2)\|\hat{\boldsymbol{w}}\|_t^2 \text{sign}(v_{\text{FedYogi}}^{t-1} - \|\hat{\boldsymbol{w}}\|_t^2). \tag{20}$$

These three adaptive optimizers have the same learning steps, including initialization, sampling subsets, and computing estimates, and they have been proved to achieve higher accuracy than FedAvg. However, these three adaptive optimizers have some differences. Unlike the FedAdagrad optimizer mainly well-suited for dealing with sparse data, both the FedAdam

and FedYogi optimizers are suitable for sparse and non-sparse data. In addition, the FedAdam optimizer can rapidly increase the learning rate, while the FedYogi optimizer increases it in a controlled fashion, for which detailed proof is provided in [118].

b) FedProx: FL has two key challenges that need to be addressed: 1) Significant variability in terms of the system characteristics of each device, which is referred to as system heterogeneity. For example, the storage, computation, and communication capabilities of each device in federated networks may be different due to variability in hardware (CPU, memory), network connectivity (3G, 4G, 5G, THz, WiFi), and power (battery level) [119], [120]. 2) Non-identically distributed data across networks, which is referred to as statistical heterogeneity [121]. Fortunately, to address these issues, FedProx has been proposed in [122] based on a federated optimization algorithm that can deal with heterogeneity both theoretically and empirically. Similar to FedAvg, FedProx randomly selects a subset of devices from K devices and averages them to form a global model. Different from FedAvg, in the local model updating of FedProx, a proximal term $\frac{\mu}{2}\|\boldsymbol{w}_g^t - \boldsymbol{w}_k^t\|^2$ is added to effectively limit the impact of variable local updates, where the local training objective of the kth device is determined by

$$\min_{\boldsymbol{w}_k^t} f_k(\boldsymbol{w}_G^t, \boldsymbol{w}_k^t) = \frac{1}{n_k}\sum_{i\in D_k} l_i(\boldsymbol{w}_k^t) + \frac{\mu}{2}\|\boldsymbol{w}_G^t - \boldsymbol{w}_k^t\|^2. \quad (21)$$

In (21), μ is the regularization parameter. There are two advantages of the proximal term purpose: 1) it addresses the issue of statistical heterogeneity by restricting the local model updating to be closer to the global model without any need to set the number of local epochs manually, and 2) it allows for the aggregation of a large number of local models resulting from system heterogeneity.

c) FOLB: Following the idea of the proximal term in FedProx, FOLB was proposed in [123]. FOLB aims at maximizing the training loss reduction in each iteration. The main difference between FedProx and FOLB is the way they update the global model. Also, FOLB can achieve higher model accuracy, training stability, and higher convergence speed over FedAvg and FedProx. Different from FedAvg and FedProx, in FOLB, in the tth round, the server selects two multisets of devices S_1^t and S_2^t with K randomly selected devices in each set, and transmits $\boldsymbol{w}_G^t$ to the kth device from set S_1^t and the $\hat{k}$th device from set S_2^t. For the kth device in S_1^t, it computes the local update $\boldsymbol{w}_k^{t+1}$, and delivers both $\boldsymbol{w}_k^{t+1}$ and loss $f_k(\boldsymbol{w}_k^t)$ to the server. For the $\hat{k}$th device in S_2^t, it only calculates and transmits its loss $f_{\hat{k}}(\boldsymbol{w}_k^t)$ to the server. Then, rather than performing simple averaging, the server calculates the global model via

$$\boldsymbol{w}_G^{t+1} = \boldsymbol{w}_G^t + \sum_{k\in S_1^t} \frac{< f_k(\boldsymbol{w}_G^t), \nabla_{S_1}\mathcal{L}(\boldsymbol{w}_G^t) >}{\sum_{\hat{k}\in S_2^t} < f_{\hat{k}}(\boldsymbol{w}_G^t), \nabla_{S_2}\mathcal{L}(\boldsymbol{w}_G^t) >} \triangle \boldsymbol{w}_k^{t+1}, \quad (22)$$

where

$$\nabla_{S_i}\mathcal{L}(\boldsymbol{w}_G^t) = \frac{1}{K}\sum_{k\in S_i^t} f_k(\boldsymbol{w}_G^t),\ i \in \{1,2\} \quad (23)$$

is the gradient of the global loss $\mathcal{L}(\boldsymbol{w}_G^t)$ obtained from the local loss of the devices in set S_i^t, and $\triangle \boldsymbol{w}_k^{t+1}$ is calculated as

$$\triangle \boldsymbol{w}_k^{t+1} = \boldsymbol{w}_k^{t+1} - \boldsymbol{w}_G^t. \quad (24)$$

In (22), the intuition is that the local update of the kth device is weighted by a measure of how correlated its gradient $f_k(\boldsymbol{w}_G^t)$ is with the global gradient $\nabla\mathcal{L}(\boldsymbol{w}_G^t)$. This correlation is assessed relative to $\nabla_{S_1}\mathcal{L}(\boldsymbol{w}_G^t)$, which is an unbiased estimate of $\nabla\mathcal{L}(\boldsymbol{w}_G^t)$ using gradient information obtained from S_1^t. The weights are normalized relative to a second unbiased estimate of the total correlation among K devices, obtained from S_2^t. Using the inner product term $< f_k(\boldsymbol{w}_G^t), \nabla_{S_1}\mathcal{L}(\boldsymbol{w}_G^t) >$ can help connect the local model to the global model, and decrease the distance between them [124]. According to [123], the comparison of the testing accuracy of FOLB, FedProx, and FedAvg on different datasets is shown in Fig. 6. It is observed that FOLB is able to achieve a higher level of testing accuracy than the other two algorithms.

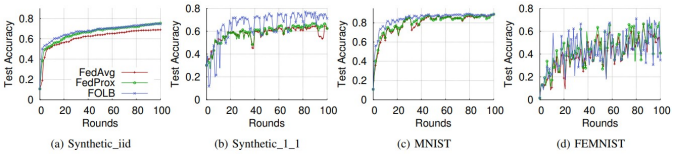


Fig. 6. Comparison of the testing accuracy of FOLB, FedProx, and FedAvg on different datasets from [123].

C. Communication Efficiency

Because of the asymmetric property of the internet connection and the large number of servers and devices, one of the major challenges in FL is the high communication overhead. There are mainly two ways to improve communication efficiency, which are decreasing and compressing the size of the learning model, respectively.

In [125], the update of the local model of the kth device is calculated as (5), and the kth device transmits $\boldsymbol{w}_k^{t+1} \in \mathbb{R}^{d_1\times d_2}$ to the server, where $\boldsymbol{w}_k^{t+1}$ has d_1 rows and d_2 columns. The authors in [125] proposed two ways to reduce the cost of transmitting local models to the server, which are structured updates and sketched updates.

For structured updates, $\boldsymbol{w}_k^{t+1}$ is limited to having a prespecified structure, either a low-rank or random-mask structure, where these two approaches are independent of each other. In the low-rank structure, $\boldsymbol{w}_k^{t+1}$ is the product of two matrices, written as

$$\boldsymbol{w}_k^{t+1} = \mathbf{A}_k^{t+1}\mathbf{B}_k^{t+1}, \quad (25)$$

where $\mathbf{A}_k^{t+1} \in \mathbb{R}^{d_1\times d}$, and $\mathbf{B}_k^{t+1} \in \mathbb{R}^{d\times d_2}$. In (25), $\mathbf{A}_k^{t+1}$ is a randomly generated matrix during the local updating, and only $\mathbf{B}_k^{t+1}$ can be optimized. Thus, $\mathbf{A}_k^{t+1}$ is presented in the form of a random seed and directly saved in the server, where the random seed means that it can generate the same random number in each time slot, and the kth device only needs to send the trained $\mathbf{B}_k^{t+1}$ to the server, which can save a factor of d_1/d during the uplink transmission. While in the random-mask structure, $\boldsymbol{w}_k^{t+1}$ is restricted to be a sparse matrix, where

the sparse matrix consists of mostly zero values. Thus, the kth device only needs to send the non-zero values of $\boldsymbol{w}_k^{t+1}$.

The sketched update is used to reduce communication costs. In this case, first the updated local model is computed without any constraints, and then the update is approximated or encoded in a compressed form before transmission to the server. Two approaches are used for the sketched update, which are subsampling and probabilistic quantizations. In subsampling quantization, rather than transmitting $\boldsymbol{w}_k^{t+1}$, the kth device only needs to transmit $\hat{\boldsymbol{w}}_k^{t+1}$ to the server, which is created from a random subset of values from $\boldsymbol{w}_k^{t+1}$. The server then averages the subsampled update and calculates the global model as (5). In probabilistic quantization, the way of compressing the local update is through quantizing each scalar of the local weights into one bit. Let $\boldsymbol{w} = (w_1, ..., w_{d_1 \times d_2}) = \text{vec}(\boldsymbol{w}_t^k)$, and let $w_{\max} = \max(w_i)$ and $w_{\min} = \min(w_i)$ $(i = 1, 2, ..., d_1 \times d_2)$, the compressed update of $\boldsymbol{w}$, denoted by $\tilde{w}$, is generated using

$$\tilde{w}_i = \begin{cases} w_{\max}, & \text{with probability } \frac{w_i - w_{\min}}{w_{\max} - w_{\min}}; \\ w_{\min}, & \text{with probability } \frac{w_{\max} - w_i}{w_{\max} - w_{\min}}. \end{cases} \tag{26}$$

According to (26), $\tilde{w}$ is an unbiased estimator of $\boldsymbol{w}$, and this method provides 32 times compression explained in [126].

A multi-objective evolutionary algorithm was designed in [127] to minimize communication costs and improve learning accuracy simultaneously. To achieve these two objectives, modified sparse evolutionary training (SET) was proposed, which can decrease the number of connections between two layers in deep neural networks (DNNs) [128]. In the modified SET algorithm, the connection probability between two layers is computed as

$$p(\bar{\boldsymbol{w}}_{ij}^k) = \frac{\varepsilon(n_k + n_{k-1})}{n_k n_{k-1}}, \tag{27}$$

and the total number of connections between two layers is calculated as

$$n_{\bar{\boldsymbol{w}}_{ij}^k} = n_k n_{k-1} p(\bar{\boldsymbol{w}}_{ij}^k), \tag{28}$$

where n_{k-1} and n_k are the number of neurons of the $(k-1)$th and the kth layer, respectively, $\bar{\boldsymbol{w}}_{ij}^k$ is the sparse weight matrix between two layers, and $\varepsilon \in (0, 1)$ is the parameter of SET to determine the connection sparsity. By setting ε, a fraction of the weights with small updates will be removed in each training epoch. In this way, the number of weights of the learning model is decreased, and thus the size of the learning model is reduced. The local and global updates still follow (6) and (7), respectively.

However, if the size of the FL model is large, decreasing or compressing the FL model via the methods in [125] and [127] still has low communication efficiency. To address this issue, quantization-based SGD has been widely adopted in FL, where the number of quantization bits and also the quantization function are optimized to minimize the total number of bits to be transmitted. Recently, sign-SGD FL was proposed in [129] to guarantee high robustness and communication efficiency. Rather than delivering gradient $\mathbf{g}_k^t$ calculated by (5), each device quantizes the gradient with a stochastic 1-bit compressor $q(.) \in \{-1, 1\}$ and sends $q(\mathbf{g}_k^t)$ to the central server. Then, the central server calculates

$$\tilde{\mathbf{g}}^t = \text{sign}(\frac{1}{K} \sum_{k=1}^{K} q(\mathbf{g}_k^t)), \tag{29}$$

and delivers $\tilde{\mathbf{g}}^t$ to the devices. The local model of each device is updated as

$$\boldsymbol{w}_k^{t+1} = \boldsymbol{w}_k^t - \eta \tilde{\mathbf{g}}^t. \tag{30}$$

In sign-FL, the size of the quantized gradient for model aggregation is just 1 bit, which significantly increases communication efficiency. However, it may lead to low learning accuracy and convergence rate. This is because fewer model weights are delivered to the central server for model aggregation, compared with the compression methods proposed in [125] and [127].

Unlike compression methods in [125], [127], and [129], federated distillation (FD) was designed in [130]. FD leverages ensemble distillation techniques and exchanges model outputs between the central server and participating devices. Therefore, in FD, each device only exchanges the output of the local model, and the communication payload size is not determined by the model size but by the output dimension, resulting in advantageous communication properties and achieving orders of magnitude reduction of the communication overhead compared with FedAvg, especially when large models are trained [131]. In FD, each device only exchanges the output of the model, the dimension of which is much smaller than that of the local model. In FD, each device treats itself as a student, and treats the averaged model output from all other devices as its teacher's output. The model output of each device is a set of logit values normalized through a softmax function, and is denoted as a logit vector whose size is determined by the number of labels of all data samples. The teacher-student output difference is measured periodically by cross entropy and becomes the loss regularizer of the student, namely, the distillation regularizer.

To guarantee communication efficiency, each device stores mean logit vectors, and periodically uploads these local-average logit vectors to a server. For each label, the uploaded local-average logit vectors from all devices are averaged, resulting in a global-average logit vector per label, which will be further downloaded to each device. When each device computes the distillation regularizer, its teacher's output is selected as the global-average logit vector associated with the same label as the current training sample's label.

In the tth time slot, the global-average logit vector $\hat{\mathbf{F}}_{k,l}^t$ is calculated as

$$\hat{\mathbf{F}}_{k,l}^t = \frac{\sum_{i=1, i \neq k}^{K} \bar{\mathbf{F}}_{k,l}^t}{K - 1}, \tag{31}$$

where $\bar{\mathbf{F}}_{k,l}^t$ is the local-average logit vector of the lth label of the kth device, and is updated as

$$\bar{\mathbf{F}}_{k,l}^t = \mathbf{F}_{k,l}^t / N_l. \tag{32}$$

In (32), N_l is the number of samples, whose learning output is the lth label, and $\mathbf{F}_{k,l}^t$ is a logit vector of the lth label calculated as

$$\mathbf{F}_{k,l}^t = \sum_{x \in \mathcal{S}_k} F_l(\boldsymbol{w}_k, x), \tag{33}$$

where $F_l(\boldsymbol{w}_k, x)$ is the logit vector of the lth label given input x and local model weights $\boldsymbol{w}_k$. In (33), $\boldsymbol{w}_k$ is still updated using (6), $\mathcal{S}_k$ is the set containing all data samples of the kth device, when the number of data samples of the kth device is larger than N_l. In the server, the local-average logit vector of the lth label $\bar{\mathbf{F}}_l^t$ of all devices is updated using

$$\bar{\mathbf{F}}_l^t = \sum_{i=1}^{K} \bar{\mathbf{F}}_{i,l}^t. \tag{34}$$

In the $(t+1)$th time slot, $\hat{\mathbf{F}}_{k,l}^{t+1}$ is updated as

$$\hat{\mathbf{F}}_{k,l}^{t+1} = \frac{\bar{\mathbf{F}}_l^t - \bar{\mathbf{F}}_{k,l}^t}{K-1}. \tag{35}$$

Then, $\hat{\mathbf{F}}_{k,l}^{t+1}$ is transmitted to the kth device. As only the logit vector is sent, the transmission size is much smaller than the learning model, allowing on-device ML to adopt large-sized local models.

D. Fairness

Most of the current FL works assume all devices contribute equally to the global model in each communication round, rather than prioritizing them based on their contributions. However, in practice, not all devices contribute equally due to various reasons, such as the different quality and quantity of the data owned by each device. Therefore, the local model from some devices may result in better global model updates, whereas those of others may impair the performance of the global model. To address this issue, fairness needs to be considered in the FL [132].

One possible learning scenario for FL in large-scale applications is that it is trained based on data originating from a large number of devices in large-scale applications, and the FL model may become biased towards certain devices. To address this issue, agnostic FL (AFL) was proposed in [133], where the global model was optimized for any target distribution formed by a mixture of device distributions, with the aim to minimize the loss function of all devices and guarantee fairness among devices. Different from FedAvg, the training objective of AFL is given by

$$\min_{\boldsymbol{w}_G} \max_{\boldsymbol{\lambda}_G} \mathcal{L}(\boldsymbol{w}_G, \boldsymbol{\lambda}_G) = \sum_{k=1}^{K} \boldsymbol{\lambda}_k f_k(\boldsymbol{w}_k), \tag{36}$$

where $\boldsymbol{\lambda}_k$ is the mixture weight of the kth device. To solve the problem, each device needs to optimize $\boldsymbol{w}_k$ and $\boldsymbol{\lambda}_k$ simultaneously. Using SGD, in the $(t+1)$th time slot, $\boldsymbol{w}_k^t$ and $\boldsymbol{\lambda}_k^t$ are calculated as

$$\boldsymbol{w}_k^{t+1} = \boldsymbol{w}_k^t - \gamma_{\boldsymbol{w}_k} \delta_{\boldsymbol{w}_k} f_k(\boldsymbol{w}_k^t), \tag{37}$$

and

$$\boldsymbol{\lambda}_k^{t+1} = \boldsymbol{\lambda}_k^t + \gamma_{\boldsymbol{\lambda}_k} \delta_{\boldsymbol{\lambda}_k} f_k(\boldsymbol{w}_k^t), \tag{38}$$

respectively, where $\delta_{\boldsymbol{w}_k} f_k(\boldsymbol{w}_k^t)$ and $\delta_{\boldsymbol{\lambda}_k} f_k(\boldsymbol{w}_k^t)$ are the unbiased estimates of the gradient, and $\gamma_{\boldsymbol{w}_k}$ and $\gamma_{\boldsymbol{\lambda}_k}$ are the respective learning rates. Then, the global model update can still be written as in (7). Using AFL, accuracy and fairness in applications with unknown mixture of device distributions can be guaranteed, thus, it can be used in large-scale networks.

A q-Fair FL (q-FFL) was proposed in [134] to encourage a fairer accuracy distribution across all devices, where q ($q > 0$) controls the tradeoff between fairness and accuracy. If $q = 0$, fairness in the FL is not encouraged. A larger q means imposing more uniformity in the training accuracy distribution and potentially inducing fairness. The objective of q-FFL is given by

$$\min_{\boldsymbol{w}_G} \mathcal{L}_q(\boldsymbol{w}_G) = \sum_{k=1}^{K} \frac{p_k}{q+1} f_k^{(q+1)}(\boldsymbol{w}_k). \tag{39}$$

To solve q-FFL in (39), it is important to first determine how to set q. In practice, q can be tuned based on the desired amount of fairness. Also, it is possible that a family of objectives with different q values has to be trained so that the algorithm can explore the trade-off between accuracy and fairness for different applications. However, one concern with addressing such a family of objectives is that it requires step-size tuning for every value of q and can cause the search space of q to explode. To solve the problem, the authors in [134] considered estimating the local Lipschitz constant, which could prevent the function value from skipping the optimal value, and dynamically adjust the step-size of the gradient-based optimization method for the q-FFL objective, avoiding manual tuning for each q [135].

The local model updates of the kth device are calculated as in (6). The global model updates are given by the sum of the first-order derivatives of $f_k^q(\boldsymbol{w}_k)$ divided by the sum of second-order derivatives of $f_k^q(\boldsymbol{w}_k)$. Thus, the kth device computes

$$\triangle \boldsymbol{w}_k^t = L(\boldsymbol{w}_G^t - \boldsymbol{w}_k^{t+1}), \tag{40}$$

$$\triangle_k^t = f_k^q(\boldsymbol{w}_k^t) \triangle \boldsymbol{w}_k^t, \tag{41}$$

$$h_k^t = q f_k^{(q-1)}(\boldsymbol{w}_k^t) \| \triangle \boldsymbol{w}_k^t \|_2 + L f_k^q(\boldsymbol{w}_k^t), \tag{42}$$

where L is the Lipschitz constant, $\triangle_k^t$ is the first-order derivative of $f_k^q(\boldsymbol{w}_k)$, and h_k^t is the second-order derivative of $f_k^q(\boldsymbol{w}_k)$. Then, the update of the global model in q-FFL is given by

$$\boldsymbol{w}_G^{t+1} = \boldsymbol{w}_G^t - \frac{\sum_{k=1}^{K} \triangle_k^t}{\sum_{k=1}^{K} h_k^t}, \tag{43}$$

where $\triangle_k^t$ and h_k^t are given in (41) and (42), respectively.

Unlike using the same version of the global model in [133] and [134], collaborative Fair FL (CFFL) was proposed in [136] to utilize reputation to update the local model of each device to converge to different models, which can achieve collaborative fairness by adjusting the performance of the models allocated to each participant based on their contributions. The reputation is applied to quantify the contribution of each device, and the reputation of the kth device is represented as

$$c_k = \sinh(\alpha * \frac{vacc_k}{\sum_{i=1}^{K} vacc_i}), \tag{44}$$

where $vacc_k$ is given as

$$vacc_k = \boldsymbol{w}_k + \eta \mathbf{g}_k. \tag{45}$$

In (44), $\sinh(\alpha)$ serves as a punishment function, and α denotes the punishment factor, that is used to distinguish the reputations of different devices based on how informative their uploaded gradients are. The larger the variation of the gradient of the local model of the kth device, the higher $vacc_k$, and the higher contribution of the kth device. The local and global models in the CFFL are updated using (6) and (7), respectively. However, the global model allocated to the kth device needs to be calculated according to its contribution using

$$\boldsymbol{w}_k^{t+1} = \frac{c_k}{\sum_{i=1}^{K} c_i} \boldsymbol{w}_G^{t+1}, \tag{46}$$

where c_k is given in (44).

Due to the fact that CFFL enables devices to converge to different final models, the most contributive device receives the most accurate model. CFFL not only can achieve comparable accuracy to FedAvg, but also can guarantee higher fairness than FedAvg.

E. Bayesian Learning

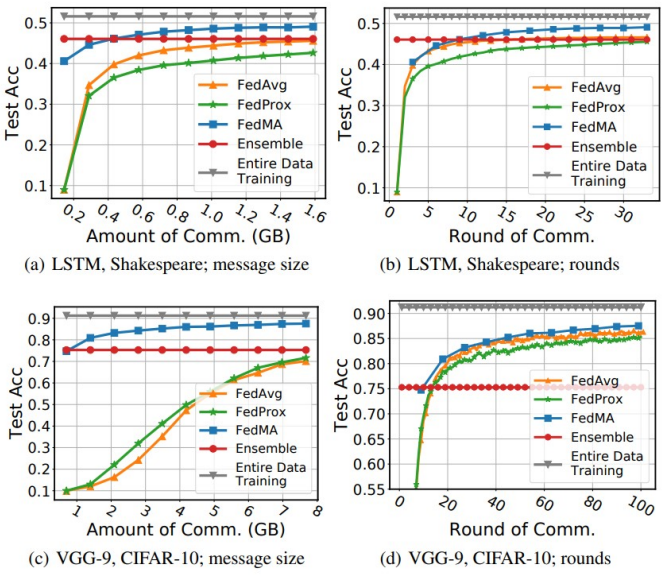
FedAvg requires access to locally stored data for learning. However, it is possible that the local model cannot be trained by the local data. Such situations may be caused by catastrophic data loss or by regulations such as the general data protection regulation [137], which places severe restrictions on the storage and access of personal data. Thus, the transmitted local model cannot be updated by the local data in the current time slot. To solve this problem, Bayesian machinery can be deployed to estimate the local model weights via probabilistic neural matching based on the pre-trained local models. Then, the local models estimated by the Bayesian machinery and the updated local models trained by local data are delivered to the server for model aggregation.

The authors in [138] proposed a probabilistic federated neural matching (PFNM) algorithm, which used a Beta Bernoulli Process (BBP) [139] to model the multi-layer perceptron (MLP) weights. Through the permutation invariance of a fully-connected neural network, the proposed PFNM algorithm first matches the weights of the transmitted local model from each device to the weights of the global model. Then, it aggregates these local models by maximizing the posterior estimation of the global weights. As a result, the PFNM algorithm can achieve higher accuracy and communication efficiency than FedAvg. However, the PFNM algorithm can only be effective for simple architectures of the neural network, such as fully-connected feedforward neural networks.

To deal with this issue, Federated Matched Averaging (FedMA) was proposed in [140], which constructed the shared global model in a layer-wise manner by matching and averaging hidden layers, including channels for convolutional neural networks (CNNs), hidden units for recurrent neural networks (RNNs), and weights for fully connected layers. The training objective of the FedMA algorithm is represented as

$$\begin{aligned} \min_{\pi_{li}^k} \sum_{i=1}^{L} \sum_{k,l} \min_{\boldsymbol{w}_{G,i}} \pi_{li}^k c(\boldsymbol{w}_{k,l}, \boldsymbol{w}_{G,i}), \\ \text{s.t.} \sum_i \pi_{li}^k = 1, \\ \sum_l \pi_{li}^k = 1, \end{aligned} \tag{47}$$

where L is the number of hidden layers, $\boldsymbol{w}_{k,l}$ denotes the weights of the lth layer learned based on the dataset of the kth device, $\boldsymbol{w}_{G,i}$ denotes the weights of the ith layer of the global model, $c(.,.)$ is an appropriate similarity function between a pair of weights, and π_{li}^k is the permutation, which determines the contribution of the weights of the lth layer of the local model of the kth device to the neurons of the ith layer of the global model. From (47), we observe that the objective is to minimize the weight distance between the local model and the global model, and $c(.,.)$ is the squared Euclidean distance. Through optimizing the permutation π_{li}^k, the total weight distance between the layers of the local model of all K devices and the layer of the global model can be minimized.



(a) LSTM, Shakespeare; message size (b) LSTM, Shakespeare; rounds

(c) VGG-9, CIFAR-10; message size (d) VGG-9, CIFAR-10; rounds

Fig. 7. Comparison of the convergence rate and testing accuracy of FedMA, FedProx, and FedAvg on different datasets from [140].

In the FedMA algorithm, first, the server gathers only the weights of the first layers from devices and performs one-layer matching to obtain the weights of the first layer of the global model. Then, the server broadcasts these weights to the devices, and each device updates the first layer of its local model to train all consecutive layers with its own dataset, keeping the matched layer frozen. The procedure continues until all layers have finished matching. Thus, FedMA requires the number of communication rounds to be equal to the number of layers in the neural network. We assume that there are N layers in the neural network. Through computing a posterior estimate (MAP) of the Bayesian non-parametric

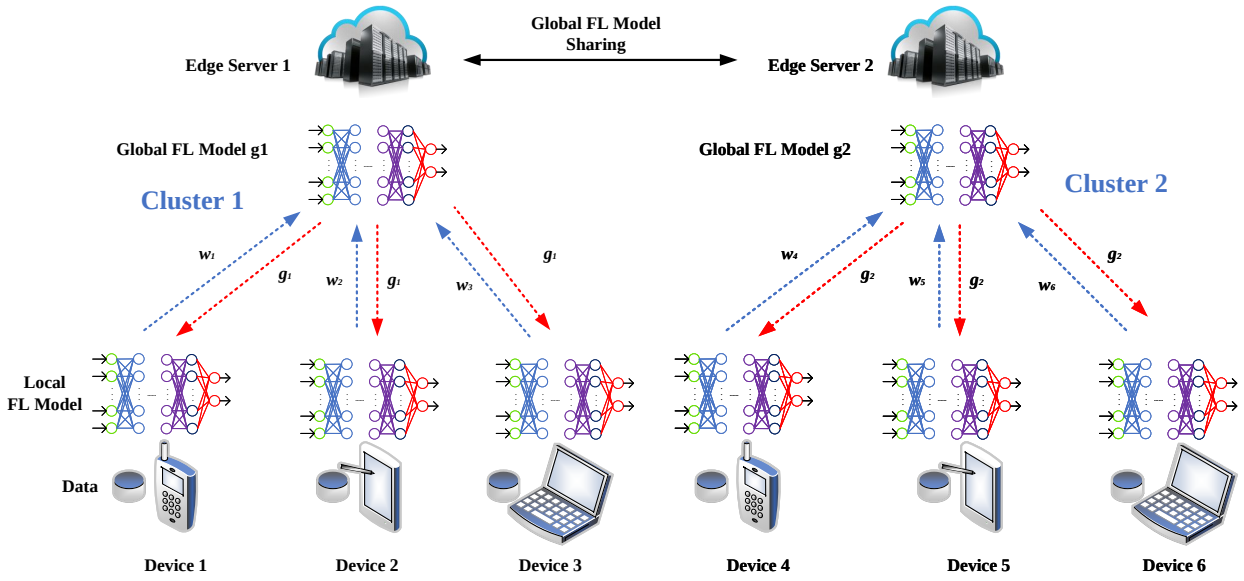


Fig. 8. Multi-client local model aggregation.

model based on the BBP, for the nth layer, if $n < N$, the global model in FedMA is updated as

$$\boldsymbol{w}_{G,n} = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{w}_{k,n}\{\prod\}_{k=1}^{K}, \tag{48}$$

where

$$\{\prod\}_{k=1}^{K} = \text{BBP-MAP}(\{\boldsymbol{w}_{k,n}\}_{k=1}^{K}). \tag{49}$$

In (49), $\{\prod\}_{k=1}^{K}$ is a permutation matrix. The permutation matrix is an orthogonal matrix and is usually used to match the layers of neural networks in the weight space. If $n = N$, the global model is updated as

$$\boldsymbol{w}_{G,N} = \sum_{k=1}^{K} p_k \boldsymbol{w}_{k,N}, \tag{50}$$

where p_k is calculated as (4). For the kth device, the local model is updated as

$$\boldsymbol{w}_{k,n+1} = \{\prod\}_k \boldsymbol{w}_{k,n+1}. \tag{51}$$

According to [140], the comparison of the convergence rate and testing accuracy of FedMA, FedProx, and FedAvg on different datasets is shown in Fig. 7. Compared to FedProx and FedAvg, FedMA not only improves communication efficiency, as the number of communication rounds is equal to the number of layers of the neural network, but also guarantees learning accuracy.

F. Clustering

For the aforementioned FL techniques, the central server updates only a single global model. In contrast, for clustered FL, the central server updates multiple global models, where the number of global models is equal to that of the clusters. Clustered FL partitions devices into different groups as in [141], where K devices were partitioned into M disjoint clusters. This method captures settings where different groups of devices have their own learning tasks. It is assumed that all devices do not have any knowledge of the other device's cluster identity. To minimize the loss function while estimating the cluster identities, an Iterative Federated Clustering Algorithm (IFCA) was proposed in [141]. In the tth time slot, the central server transmits M updated global models to these M clusters. Then, a random subset of devices is selected to update their local models with their corresponding data samples and delivers the local models to the central server. As the central server does not know the cluster identities of the selected devices, it estimates the identity of the kth device using

$$\hat{j} = \arg\min_{j\in\mathcal{M}} f_k(\boldsymbol{w}_{G,j}^t), \tag{52}$$

where $\mathcal{M}$ is the set including all clusters, $f_k(\boldsymbol{w}_{G,j}^t)$ is the loss of the kth device in the jth cluster, and $\boldsymbol{w}_{G,j}^t$ is the global model of the jth cluster. From (52), we observe that the kth device belongs to the $\hat{j}$th cluster that achieves the minimum loss. Given the estimated clusters, the global model of each cluster is updated using (7), and the local model of each device is still updated using (6).

However, in [141], all selected devices need to communicate with the central server, thus, there is a large overhead when a large number of devices transmit. To address this issue, a hierarchical clustering (HC) algorithm for local model updating was proposed in [142], where the devices were iteratively merged into clusters with high simmilarity by calculating L1 (Manhattan), L2 (Euclidean), and cosine distance metrics. In the tth time slot, the d-norm distances between all clusters are calculated to judge their similarity. The distance $\hat{d}_{i,j}$ between the ith cluster and the jth cluster in the global model is calculated as

$$\hat{d}_{i,j} = \|\boldsymbol{w}_{G,i}^t - \boldsymbol{w}_{G,j}^t\|_d. \tag{53}$$

If $\hat{d}_{i,j}$ is smaller than the threshold $\hat{d}_{\text{th}}$, the ith and jth clusters can be merged together. This procedure continues until all

clusters with similarity are merged into a single cluster. Then, these merged clusters select a portion of their devices to aggregate the local models in the server via (7), and the local models in the devices are still updated using (6). By selecting a portion of the devices in the merged clusters to aggregate the local models, the communication overhead is reduced.

Although authors in [141] and [142] considered clustered FL, both works adopted a single central server to capture the global models of all devices by aggregating their local models. In [143], a multi-center aggregation mechanism for multiple global models in clustered FL was proposed, where devices belong to a specific cluster, and the cluster updates its own global model with its corresponding updated local models, as shown in Fig. 8. Each device calculates the distance between its local model and the global model of the server in each cluster. The device selects the cluster with the minimum distance. The learning objective of multi-center clustered FL is to minimize the total weighted distance between the global model and the local models, and the multi-center weight distance-based loss (MD-Loss) is represented as

$$\mathcal{L} = \frac{1}{K}\sum_{i=1}^{m}\sum_{k=1}^{K} r_{k,i}\ \mathrm{Dist}(\boldsymbol{w}_k, \boldsymbol{w}_{G,i}), \tag{54}$$

where

$$\mathrm{Dist}(\boldsymbol{w}_k, \boldsymbol{w}_{G,i}) = \|\boldsymbol{w}_k - \boldsymbol{w}_{G,i}\|_2, \tag{55}$$

$r_{k,i} = \{0, 1\}$ is the cluster assignment, $r_{k,i} = 1$ indicates that the kth device belongs to the ith cluster, vice versa. In (54), K and m are the number of devices and clusters, respectively. In (55), $\boldsymbol{w}_{G,i}$ is the global model of the ith cluster. To solve (54), a federated stochastic expectation maximization (FeSEM) algorithm is deployed in the following three steps:

First, the cluster assignment $r_{k,i}$ is updated using

$$r_{k,i} = \begin{cases} 1, & \text{if } i = \arg\min_j \mathrm{Dist}(\boldsymbol{w}_k, \boldsymbol{w}_{G,j}); \\ 0, & \text{otherwise.} \end{cases} \tag{56}$$

From (56), we observe that the kth device belongs to the ith cluster that can achieve the minimum weight distance.

Second, the global model of the kth cluster is aggregated as

$$\boldsymbol{w}_{G,i} = \frac{1}{\sum_{k=1}^{m} r_{k,i}} \sum_{k=1}^{K} r_{k,i}\boldsymbol{w}_k. \tag{57}$$

Finally, the local model of the kth device is still updated using (6). The multi-center aggregation mechanism can better capture the heterogeneity of data distributions across devices, and simultaneously facilitates the optimal matching between devices and servers.

FL with a single server or multiple servers still has high vulnerability when there exists a failure or an attack on the server. Therefore, unlike authors in [141]–[143] considered single or multiple servers for model aggregation, authors in [144] proposed a general fully-decentralized FL framework where the device was denoted as a "server" that update its local model by aggregating models from other devices. The model updating follows (7). Furthermore, authors in [145] derived convergence analysis of fully decentralized FL under the condition that devices update their local models by aggregating models from their one-hop neighbors.

G. Convergence Analysis of FL Methodologies

The derivation of convergence analysis of FL is usually based on the following fundamental assumptions.

Assumption 1. (Lipschitz). Loss functions $f_1, ..., f_N$ are all $\rho-$ lipschitz for any $\boldsymbol{w}$ and $\boldsymbol{w}^{'}$:

$$\|f_n(\boldsymbol{w}) - f_n(\boldsymbol{w}^{'})\| \leq \rho\|\boldsymbol{w} - \boldsymbol{w}^{'}\|, \tag{58}$$

where ρ is a positive constant.

Assumption 2. (Smoothness). Loss functions $f_1, ..., f_N$ are all $L-$ smooth for any $\boldsymbol{w}$ and $\boldsymbol{w}^{'}$:

$$\|\nabla f_n(\boldsymbol{w}) - \nabla f_n(\boldsymbol{w}^{'})\| \leq L\|\boldsymbol{w} - \boldsymbol{w}^{'}\|, \tag{59}$$

where L is a positive constant.

Assumption 3. (Bounded Gradient and model). The second moments of stochastic gradients and weights are bounded for any $\boldsymbol{w}$, and they are guaranteed by the l_2-regularization [146], [147], which are denoted as

$$\mathbb{E}\|\nabla f_n(\boldsymbol{w})\|^2 \leq G, \tag{60}$$

and

$$\mathbb{E}\|\boldsymbol{w}\|^2 \leq A. \tag{61}$$

In (60) and (61), both G and A are positive constants.

Assumption 4. (Gradient divergence). The gradient divergence for any n and $\boldsymbol{w}$ is denoted as

$$\mathbb{E}\|\nabla f_n(\boldsymbol{w}) - \nabla f(\boldsymbol{w})\|^2 \leq \hat{\sigma}^2. \tag{62}$$

In (62), $\hat{\sigma} > 0$ is a constant.

For FL methodologies, researchers usually derive the number of communication rounds or the upper bound of $\mathbb{E}\|\nabla f(\boldsymbol{w})\|^2$ in convergence analysis. We will give several examples as follows.

In SCAFFOLD, based on assumptions 2 and 3, the number of communication rounds to achieve convergence in non-convex FL settings is derived as

$$R = \mathcal{O}\left(\frac{L\hat{\sigma}^2(f(\boldsymbol{w}^0) - f(\boldsymbol{w}^*))}{NS\epsilon} + \left(\frac{K}{S}\right)^{\frac{2}{3}}\frac{B(f(\boldsymbol{w}^0) - f(\boldsymbol{w}^*))}{\epsilon}\right), \tag{63}$$

where N is the number of local updates, K is the total number of devices, S is the selected number of devices for model aggregation, and ϵ is the required learning accuracy to reach.

In FedProx, based on assumptions 1 and 3, after $T = \mathcal{O}(\frac{f(\boldsymbol{w}^0) - f(\boldsymbol{w}^*)}{\rho G})$ iterations, it is obtained that

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\|\nabla f(\boldsymbol{w}^t)\|^2] \leq G. \tag{64}$$

Remark: Although most research works consider the same assumptions for convergence analysis, there is no uniform equation in convergence analysis for all FL algorithms. Therefore, the convergence rate for different FL settings should be derived according to specific situations in detailed FL works.

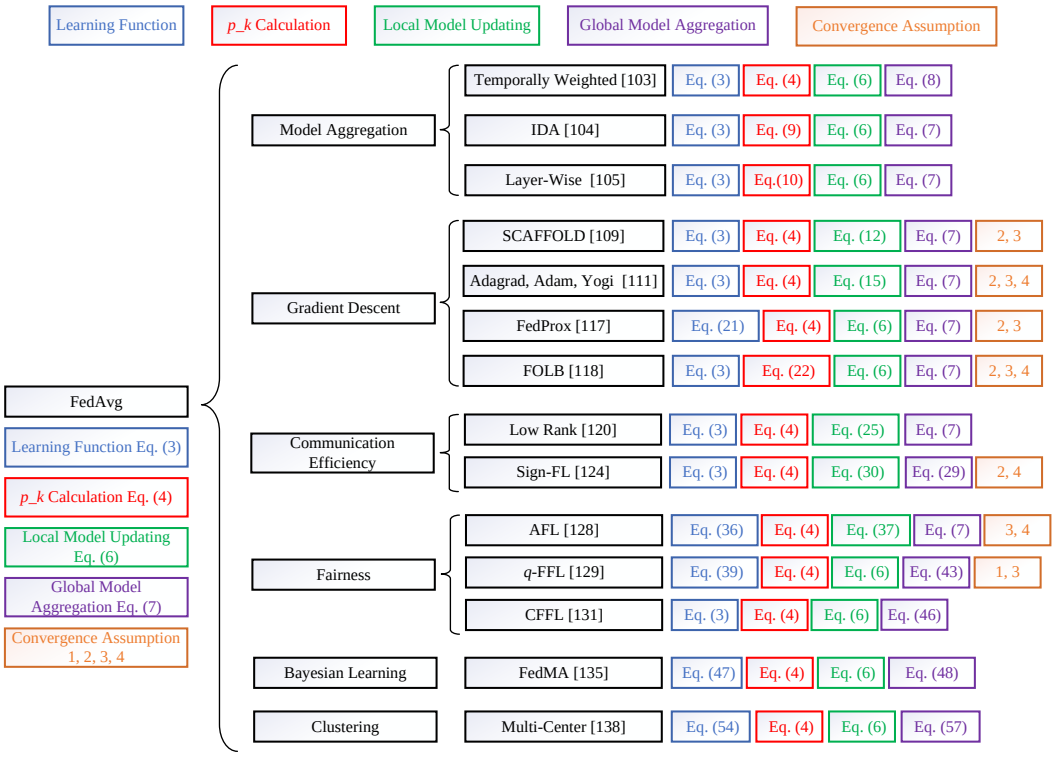


Fig. 9. Evolution of FedAvg in FL methodologies.

H. Summary and Lessons Learned

In this section, we have reviewed six main research areas of FL methodologies. We summarize the approaches along with references. From this review, we gather the following lessons learned:

- The differences between FedAvg and other FL algorithms mentioned are presented in Fig. 9. Based on Fig. 9, FedAvg includes four components, which are the learning function, p_k calculation, local model updating, and global model aggregation, corresponding to equations (3), (4), (6), and (7), respectively. Through jointly or separately optimizing these four equations, the learning efficiency, accuracy, and fairness can be improved. In addition, theoretical convergence analysis is usually derived based on four fundamental assumptions.
- For the research works we have discussed in this section, the authors assumed that the proposed FL algorithms are implemented in a synchronous mode. The heterogeneity among devices, such as computation capabilities, is usually not considered. Meanwhile, authors assume that all devices successfully transmit and receive local models and updated global models. In real-world applications, the approach may not be feasible for devices with weak processing power or unstable network connection, which can lead to straggler effect.
- The simulations in the research works we have discussed are usually performed on MNIST, FashionMNIST, and CIFAR. The data in these datasets are usually labeled and the data distribution of them is clear. Therefore, whether the proposed FL algorithms are effective in unlabeled datasets or other datasets with unknown data distribution still needs to be investigated.
- For the studies that we have discussed in this section, local models are trained with non-poisoned data. Data poisoning refers to the malicious device who either actively or passively uses some poisoned data for model training, and model poisoning means that an attacker tampers the weights of the local model, which further affects the parameters of the global model. Although FL by itself has a certain level of resilience against attacks, the frequent communications between server and devices may spread the risk over the networks and thus breaks down the overall learning systems. Therefore, how to design a FL algorithm that is robust to poisoned data and design resilient networks for FL to avoid spreading attacks needs to be investigated.

IV. FL in Wireless Networks

Recently, there is a growing interest in optimizing wireless networks with data-driven ML-based methods. In this section, based on the fundamental FL algorithms in Section III, we present the research areas of designing FL for wireless networks. Considering FL in wireless networks, all weights of local or global models are delivered via wireless links instead of wired ones. Thus, the model aggregation, learning accuracy, and learning efficiency of FL can be influenced by wireless factors, such as the set of devices that participate in FL, computational capacity, transmission power and wireless channel, and spectrum resource allocation. The impact of these factors on FL performance is introduced as follows:

- With the increasing number of devices participating in model aggregation, the generalization of the global model increases. However, more devices lead to high interference and a low transmission rate.
- Computational capacity of each device affects the learning latency. High computational capability leads to high learning efficiency and low learning latency.
- Transmission power and wireless channel determine transmission rate and reliability. Low transmission power and dynamic wireless channel result in low transmission rates and high transmission errors.
- Spectrum resource allocated to each device affects transmission rate. When more spectrum is allocated to the device, its transmission rate increases.

To exchange a large number of model weights over time-varying channels, there are two types of solutions, which are "digital" and "analog" approaches that convert all global or local models into bits and modulated signals, respectively. In this section, we introduce these key research areas of FL over wireless networks. For the digital approach, we consider model aggregation, communication, energy, and computation efficiency optimization, resource allocation, and asynchronous FL. For the analog approach, we consider over-the-air computation.

A. Digital Approach of FL over Wireless Networks

The digital approach needs to guarantee a high transmission rate, low transmission error, and high communication, energy, and computation efficiency of FL over wireless networks. In this subsection, we mainly introduce 1) Model Aggregation, 2) Communication, Energy, and Computation Efficiency Optimization, 3) Resource Allocation, and 4) Asynchronous FL.

1) Model Aggregation: The model uploading and downloading of FL can be affected by dynamic wireless channels. Due to the limited bandwidth of the wireless network, not all devices can transmit local models to the central server. Also, poor channel state results in high transmission error. To select optimal devices to upload the local models and minimize transmission error, device selection and packet transmission error minimization schemes need to be considered.

Device Selection: Three main device selection schemes have been studied, which are probabilistic updating, importance-based updating, and novel communication protocol-based updating.

a) Probabilistic Updating: A traditional probabilistic scheduling policy was developed in [148] to characterize the convergence performance of FL in wireless networks. In particular, the effectiveness of three different scheduling policies, i.e., random scheduling (RS), round robin (RR), and proportional fair (PF) were considered to select a portion of devices for local model aggregation under limited bandwidth constraints. In RS, the access point (AP) randomly selects K associated devices in each time slot for local model updating, and each device is allocated with a sub-channel to deliver the local model. In RR, the AP arranges all devices into G groups and assigns each group to access the radio channels and update their weights in each time slot. While in PF, the AP selects K out of $\hat{K}(K \leq \hat{K})$ associated devices in each time slot according to the following policy

$$\mathbf{m}^* = \underset{\mathbf{m}\in\{1,2,...,\hat{K}\}}{\arg\max} \left\{ \frac{\tilde{\rho}_{1,t}}{\bar{\rho}_{1,t}}, ..., \frac{\tilde{\rho}_{K,t}}{\bar{\rho}_{K,t}} \right\}, \tag{65}$$

where $\mathbf{m} = \{m_1, ..., m_K\}$ is a length-K vector and $\mathbf{m}^* = \{m_1^*, ..., m_K^*\}$ represents the indices of the selected K devices, $\tilde{\rho}_{k,t}$ and $\bar{\rho}_{k,t}$ are the instantaneous and time average signal-to-noise ratio (SNR) of the kth device in the tth time slot, respectively. Thus, the device with a higher SNR is selected [149]. The updating of the local and global models still follows (6) and (7), respectively.

Based on the probabilistic analysis of the scheduling policies in [148], it shows that PF outperforms RS and RR in terms of convergence rate under a high SNR threshold. This is because a high SNR threshold reduces the chance of successful transmission from an arbitrary device, while PF improves the convergence rate by selecting devices with better channel qualities in order to increase their transmission success probabilities. However, RR is preferable when the SNR threshold is low, this is because low SNR threshold results in a high success probability.

b) Importance-based Updating: To exploit the importance of devices, a novel probabilistic scheduling framework was developed to apply unbiased update aggregation for the federated edge learning (FEEL) in [150], where the importance of a local model update was measured by its gradient divergence. In the tth time slot, K devices are selected for model aggregation according to a scheduling distribution $\mathcal{P}^t = (p_1^t, p_2^t, ..., p_K^t)$, where p_k^t is the probability that the kth device is selected, and can also indicate the level of importance that the kth device can contribute to the global model convergence. The local model is still updated using (6), and the global model is updated using

$$\boldsymbol{w}_G^t = \frac{1}{n} \sum_{k=1}^{K} \frac{n_k^t}{p_k^t} \boldsymbol{w}_k^t, \tag{66}$$

where n_k is the number of data at the kth device, and $n = \sum_{i=1}^{K} n_i$. The selected local model $\boldsymbol{w}_k^t$ needs to be scaled by a coefficient $\frac{n_k^t}{n}$ at the edge server. This is because this coefficient well quantifies the unbalanced property in global data distribution and thus makes the global model unbiased. Based on the probabilistic scheduling framework, the importance indicator of each local model is defined as

$$I_k^t = \left\| \frac{n_k^t}{n p_k^t} \boldsymbol{w}_k^t - \boldsymbol{w}_G^t \right\|_2. \tag{67}$$

The model divergence reflects the deviation between the local model and the global model, and the smaller the model divergence, the more it can contribute to the global model convergence. In other words, the smaller the p_k^t, the less contribution of the kth device to the global model. The optimization problem is to achieve a trade-off between gradient divergence and latency, where $\rho \in [0, 1]$ is defined as the

weight coefficient that balances the gradient divergence and latency. Then, the objective function is represented as

$$\min_{(p_1^t,\ldots,p_K^t)} \sum_{k=1}^{K} p_k^t \left[\rho I_k^t + (1-\rho) T_k^t\right], \tag{68}$$

$$\text{s.t.} \sum_{k=1}^{K} p_k^t = 1, \tag{69}$$

$$p_k^t \geq 0, \tag{70}$$

where I_k^t is given by (67), and the optimal probability p_k^{t*} is given by

$$p_k^{t*} = \frac{n_k}{n} \|\boldsymbol{w}_k^t\| \sqrt{\frac{\rho}{(1-\rho)T_k^t + \lambda^t}}, \tag{71}$$

where λ^t is the lagrangian multiplier that satisfies (69), and T_k^t is the uplink transmission latency. From (71), the optimal scheduling decision is mainly determined by data unbalanced indicator $\frac{n_k}{n}$, the norm of local model $\|\boldsymbol{w}_k^t\|$, and the uplink latency T_k^t. Through the importance-aware device scheduling strategy, it can achieve less than half of the convergence time and up to 2% higher final accuracy.

c) Novel Communication Protocol-based Updating: Apart from deploying probabilistic scheduling to select the optimal devices for model aggregation, a communication protocol designed for FL over wireless networks, called federated learning with client selection (FedCS), was proposed in [151]. First, the server requests $\lceil K \times C \rceil$ random devices to participate in the current training task, where K is the total number of all devices, $C \in (0, 1]$ is the fraction of devices that participating in training in each time slot, and $\lceil . \rceil$ is the ceiling function. The server selects as many devices as possible to transmit their local models for model aggregation within a certain deadline after receiving the resource information, such as wireless channel states, computational capacities, and size of data resources, from devices. The optimization problem is to maximize the number of devices for model aggregation as

$$\max \quad S, \tag{72}$$

$$\text{s.t. } T_{\text{CS}} + T_{\text{Agg}} + T_{\text{up}} + T_{\text{down}} \leq T_{\text{th}}, \tag{73}$$

where S is the number of the selected devices for model aggregation, and T_{CS}, T_{Agg}, and T_{down} are the time required for device selection, model aggregation, and downlink transmission, respectively. In (73), T_{up} is the time required for local model updating and uplink transmission.

To solve the optimization problem in (72), a heuristic algorithm based on the greedy algorithm was proposed. Although the proposed FedCS algorithm can achieve a higher accuracy than that of FedAvg, the computation complexity is extremely high with a large number of devices. Also, the local and global updating still follow (6) and (7), respectively.

However, authors in [148], [150], [151] considered perfect wireless channels, and in the uplink transmission, there will be transmission errors caused by unstable wireless channels. To solve the problem, the impact of the packet transmission error on FL is considered.

Packet Transmission Error: The authors in [152] considered packet transmission errors, which could affect the local model aggregation of FL. In [152], a closed-form solution for the convergence rate of FL was derived as a function of packet error rates. Based on this solution, the BS optimizes the resource allocation and the device optimizes its transmission power to decrease packet error rates. The optimization problem in [152] is to minimize the training loss of FL over wireless networks. The expression of the model aggregation with packet transmission error is denoted as

$$\boldsymbol{w}_g(\mathbf{a}^t, \mathbf{P}^t, \mathbf{R}^t) = \frac{\sum_{k=1}^{K} \bar{N}_k a_k^t \boldsymbol{w}_k^t C(\boldsymbol{w}_k^t)}{\sum_{k=1}^{K} \bar{N}_k a_k^t C(\boldsymbol{w}_k^t)}. \tag{74}$$

In (74), $\sum_{k=1}^{K} \bar{N}_k a_k^t C(\boldsymbol{w}_k^t)$ is the total number of training data samples, which is determined by a_k^t and $C(\boldsymbol{w}_k^t)$, N_k is the number of training data samples of the kth device, $a_k^t \in \{0, 1\}$ is the device association index of the kth device. If $a_k^t = 1$, the kth device is selected to update the local model to the BS, and vice versa. Also, in (74), $\mathbf{P}^t = [P_1^t, ..., P_K^t]$ is the transmit power matrix, $C(\boldsymbol{w}_k^t)$ is the packet transmission index of the kth device, which is presented as

$$C(\boldsymbol{w}_k^t) = \begin{cases} 1, & \text{with probability } 1 - q_k(r_k^t, P_k^t), \\ 0, & \text{with probability } q_k(r_k^t, P_k^t), \end{cases} \tag{75}$$

and $q_k(r_k^t, P_k^t)$ is the packet error rate of the local model of the kth device. If $C(\boldsymbol{w}_k^t) = 0$, the local model of the kth device contains data error, and the BS will not use it to update the global model. In (75), $r_{k,n}^t$ is the resource block (RB) allocation index, and $r_{k,n}^t = 1$ means that the nth RB is allocated to the kth device in the tth time slot. Meanwhile, in (75), $q_k(r_k^t, P_k^t)$ is expressed as

$$q_k(r_k^t, P_k^t) = \sum_{n=1}^{R} r_{k,n}^t q_{k,n}^t, \tag{76}$$

where $q_{k,n}^t$ is the packet error rate over the nth RB with m being a waterfall threshold and is defined as

$$q_{k,n}^t = E_{h_k}\left(1 - \exp\left(-\frac{m(\bar{I}_n + BN_0)}{P_k^t h_k}\right)\right). \tag{77}$$

In (77), h_k is the channel gain between the BS and the kth device, N_0 is the noise power spectral density, and $\bar{I}_n$ is the interference caused by the other devices. The local model updating is still written as (7). Through optimizing the power and RB allocation via the Hungarian algorithm [153], both the packet transmission error and the training loss can be minimized.

2) Communication, Energy and Computation Efficiency Optimization: For FL over wireless networks, one of the challenges is to maximize the communication, energy, and computation efficiency, which can be influenced by the bit rate, energy, and computation capability. In this subsection, the research on the optimization of communication, energy, and computation efficiency is introduced.

a) Communication Efficiency: Authors in [154] introduced momentum gradient and sparse communication to increase the communication efficiency of FL over wireless networks. To optimize the transmission rate of each device, the optimal sub-carrier is allocated to each device. The modified momentum method is used to accelerate the performance of

SGD. Then, based on the sparse communication, the global model is updated using

$$\boldsymbol{w}_G^{t+1} = \sum_{k=1}^{K} p_k \text{sparse}(\boldsymbol{w}_k^t), \tag{78}$$

where p_k is calculated by (4), and sparse function $sparse()$ in (78) converts $\boldsymbol{w}_k^t$ to sparse form by squeezing out any zero elements [155]. Using sparse communication, the server and devices only transmit a fraction of the weights that considerably reduce the communication latency. With the help of momentum and sparse communication, the convergence speed and latency of FL over wireless networks can be guaranteed, however, the accuracy decreases.

It is important to know that training and transmitting weights during FL may consume a large amount of energy. To deal with this issue, the energy consumption minimization problem was studied.

b) Energy Efficiency: To minimize the total energy consumption for local computation and wireless transmission, an iterative algorithm was proposed in [156]. The total energy consumption of all devices at each step is calculated as

$$E = \sum_{k=1}^{K} (E_k^C + E_k^T), \tag{79}$$

where E_k^C and E_k^T are the local computation energy and wireless transmission energy of the kth device, respectively. To minimize the energy, closed-form solutions for the time allocation t_k, bandwidth allocation b_k, power control p_k, computation frequency f_k, and learning accuracy δ are derived. At each iteration, to optimize $(t_k, b_k, p_k, f_k, \delta)$, the authors first optimized (t_k, δ) under fixed (b_k, p_k, f_k). Then, (b_k, p_k, f_k) are updated based on the obtained (t_k, δ). Thus, the optimal solution of (b_k, p_k, f_k) or (t_k, δ) can be obtained at each time slot. Also, the local and global model updatings still follow (6) and (7), respectively.

The works in [154] and [156] mainly focused on accelerating the training tasks from the communication and energy perspective, i.e., minimizing the communication overhead and energy consumption. However, computation efficiency is also one of the major characteristics of FL over wireless networks, which may greatly affect learning performance.

c) Computation Efficiency: There are mainly two ways to increase computation efficiency for FL in wireless networks, including deploying high computation units and efficient gradient descent methods.

High Computation Unit: In recent years, GPU has been proposed to accelerate the training latency and efficiency of FL. Authors in [157] considered the training acceleration problem from the CPU to GPU under communication and computation resource constraints. Using the Karush-Kuhn-Tucker (KKT) conditions, the closed-form solutions of joint batch size selection and communication resource allocation were derived, and the relationship between training latency and training batch size was analyzed. The local and global updatings in [157] still follow (6) and (7), respectively. Although using GPU for FL training improved the learning efficiency, authors in [157] relied on an impractical assumption that each device was equipped with a GPU. The mobile device with GPU for training can cost a large amount of energy, especially for battery-limited devices.

Gradient Descent: To improve the training performance of FL over wireless networks, several novel gradient descent methods have been studied. For the gradient descent methods in FL over wireless networks, the authors in [158] and [154] mainly deployed SGD and momentum gradient descent to update the local model, respectively.

(1) SGD: Different from the gradient descent in (6), that calculated from the entire dataset, SGD randomly selects one data sample from the whole dataset at each time slot to reduce computation complexity [159]. For the SGD in [158], the local model of the kth device in the tth time slot is updated as

$$\boldsymbol{w}_k^{t+1} = \boldsymbol{w}_k^t - \eta \mathbf{g}_{k,i}^t, \tag{80}$$

where i is the ith data sample of the kth device.

(2) Momentum Gradient Descent: Momentum is a method that helps to accelerate the gradient descent in the relevant direction and dampens oscillations. This is achieved by adding a momentum term σ of the update vector of the past time slots to the current update vector [160]. For the gradient descent with momentum in [154], the local model is updated using

$$\boldsymbol{w}_k^t = \boldsymbol{w}_k^{t-1} + \mathbf{u}_k^t, \tag{81}$$

where

$$\mathbf{u}_k^t = \sigma \mathbf{u}_k^{t-1} + \mathbf{g}_k^t. \tag{82}$$

Also, the model aggregation of both gradient descent methods follows (7). Using SGD and momentum, a faster convergence rate can be achieved. Note that the gradient descent methods of FL methodologies mentioned in Section III can still be deployed in FL over wireless networks.

Authors in [154], [156]–[158] optimized the communication, energy, and computation efficiency, separately. In fact, the communication, energy, and computation efficiencies are correlated with each other, which requires joint design and optimization among them.

d) Joint Design of Communication, Energy, and Computation Efficiency: In FL over wireless networks, there are two trade-offs, between computation and communication latencies, and between learning latency and device energy consumption. Authors in [161] decomposed the problem into two sub-problems, which were the learning latency versus device energy consumption problem solved by the Pareto efficiency model, and the computation versus communication latencies problem, solved via finding the optimal learning accuracy with KKT conditions. The Pareto efficiency model minimizes the learning latency and does not increase the energy costs of each device [162]. Through iteratively obtaining the closed-form solutions of these two sub-problems, the authors characterized how the computation and communication latencies of mobile devices affect trade-offs between energy consumption, learning time, and learning accuracy. However, authors in [161] relied on an impractical assumption that the channel state information (CSI) remained unchanged during the whole FL process. In [163], the authors considered imperfect CSI. Under imperfect CSI, channels between the server and devices

over each resource block (RB) are predicted using their past observations. Then, based on Lyapunov optimization, joint device scheduling and RB allocation policy were proposed to minimize the loss function in FL over wireless networks. The local and global model updatings in [161], [163] still follow (6) and (7), respectively.

In summary, authors in [148], [150]–[152], [154], [156]–[158], [161], [163] independently considered the model aggregation and communication, energy, and computation efficiency optimization. However, in practical scenarios, they are correlated with each other, and in the next section, we introduce the resource allocation of FL over wireless networks.

3) Resource Allocation: For the research on resource allocation of FL over wireless networks, it jointly designs the spectrum resource allocation and device selection. Due to the limited number of RBs in the uplink transmission, only a fraction of devices can transmit local models to the base station (BS). To solve this problem, a probabilistic device selection scheme was proposed in [164] to select the devices whose local models have significant effects on the global model. The update of the global model in the tth time slot is still updated as (7), while p_k in (7) is calculated as

$$p_k = \frac{a_k^t \bar{N}_k}{\sum_{i=1}^{K} a_i^t \bar{N}_i}, \tag{83}$$

where $a_k^t \in \{0,1\}$ is the device association index of the kth device, and $\bar{N}_k$ is the number of training data samples of the kth device. If $a_k^t = 1$, the kth device is selected to update the local model to the BS, vice versa. To select the optimal set of devices to upload the local models and minimize the uplink and downlink transmission latency, the optimization problem of a joint RB allocation and device selection scheme was proposed and written as

$$\min_{\mathbf{A},\mathbf{R}} \sum_{t=1}^{\hat{C}} T^t(\mathbf{a}^t, \mathbf{R}^t) I^t \tag{84}$$

$$\text{s.t. } a_k^t, r_{k,n}^t, I^t \in \{0,1\}, \tag{85}$$

$$\sum_{t=1}^{K} r_{k,n}^t \leq 1, \tag{86}$$

$$\sum_{n=1}^{R} r_{k,n}^t = a_k^t, \tag{87}$$

where $T^t(\mathbf{a}^t, \mathbf{R}^t)$ is the transmission latency, $\mathbf{A} = [\mathbf{a}_1, ..., \mathbf{a}_{\hat{C}}]$ is the device selection matrix of all iterations, $\mathbf{R} = [\mathbf{R}_1, ..., \mathbf{R}_{\hat{C}}]$ is the RB allocation matrix for all devices of all iterations, R is the number of RBs, and $\hat{C}$ is a constant, which is large enough for the proposed FL to converge. In (85), $I^t = 0$ means that the proposed FL converges, vice versa. In (86), $r_{k,n}^t = 1$ means that the nth RB is allocated to the kth device in the tth time slot, and (86) implies that at most one RB is allocated to the kth device. In (87), it means that all RBs should be allocated to the devices associated with the BS. To increase the convergence speed of FL, deep neural networks (DNNs) are used to predict the local models of devices that cannot transmit their local model weights. To enable the BS to predict the local model, each device should have a chance of connecting to the BS to provide local model weights for training DNNs. Thus, a probabilistic device association scheme was proposed as

$$\bar{P}_k^t = \begin{cases} \frac{\|\mathbf{e}_k^t\|}{\sum_{k=1,k\neq k^*}^{K} \|\mathbf{e}_k^t\|}, & \text{if } k \neq k^*, \\ 1, & \text{if } k = k^*, \end{cases} \tag{88}$$

where $\bar{P}_k^t$ is the probability of the kth device connecting to the BS in the tth time slot, $\mathbf{e}_k^t = \boldsymbol{w}_G^t - \boldsymbol{w}_k^{t+1}$ is the variation between the global model and the local model of the ith device, and $\|\mathbf{e}_k^t\|$ is the norm of $\mathbf{e}_k^t$. In (88), the association probability between the BS and the kth device increases as $\|\mathbf{e}_k^t\|$ increases. Thus, the probability that the BS deploys the local model of the kth device to aggregate the global model increases. Through using the device association scheme in (88), the BS has a higher probability of selecting devices whose local models significantly affect the global model. In addition, the k^*th device is always connected to the BS to provide a local model for the prediction of other devices' local models. With the predicted local models, the global model is updated as

$$\boldsymbol{w}_G^{t+1} = \frac{\sum_{k=1}^{K} \bar{N}_k a_k^t \boldsymbol{w}_k^t + \sum_{k=1}^{K} \bar{N}_k (1 - a_k^t) \hat{\boldsymbol{w}}_k^t \mathbb{I}_{\{E_k^t \leq \gamma\}}}{\sum_{k=1}^{K} \bar{N}_k a_k^t + \sum_{k=1}^{K} \bar{N}_k (1 - a_k^t) \mathbb{I}_{\{E_k^t \leq \gamma\}}}, \tag{89}$$

where $\hat{\boldsymbol{w}}_k^t$ is the predicted local model of the kth device, $\sum_{k=1}^{K} \bar{N}_k a_k^t \boldsymbol{w}_k^t$ is the sum of local models of the devices connected to the BS, $\sum_{k=1}^{K} \bar{N}_k (1 - a_k^t) \hat{w}_k^t \mathbb{I}_{\{E_i^k \leq \gamma\}}$ is the sum of the predicted local models of the devices are not connected to the BS, $E_k^t = \|\hat{\boldsymbol{w}}_k^t - \boldsymbol{w}_k^t\|^2$ is the prediction error, and γ is the error threshold. If $E_k^t \leq \gamma$, the BS uses the predicted local model $\hat{\boldsymbol{w}}_k^t$ to update the global model, otherwise, not. From (89), we can obtain that the BS uses the predicted local models together with the transmitted local models to update the global model to decrease the FL training loss and improve the convergence speed.

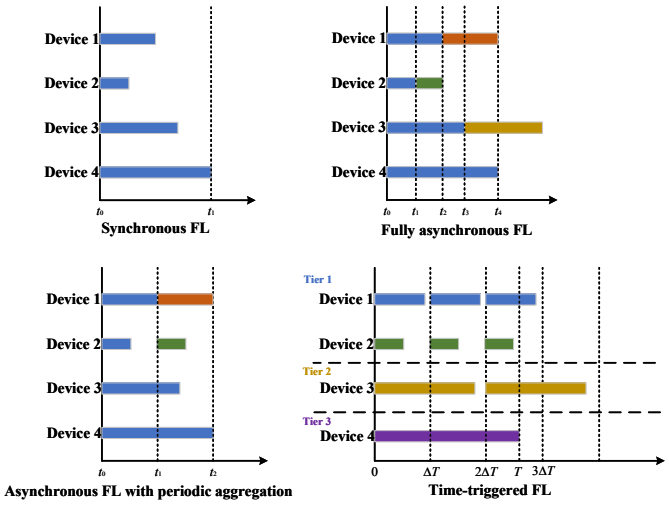


Fig. 10. Illustration of synchronous FL, fully asynchronous FL, asynchronous FL with periodic aggregation, and time-triggered FL.

4) Asynchronous FL: Authors in [148], [150]–[152], [154], [156]–[158], [161], [163], [164] mainly considered synchronous FL over wireless networks. One common problem of FL systems is the straggler issue. This problem originates from the fact that the time duration of each training round is strictly limited by the slowest participating device [165]. Two asynchronous FL policies are introduced in [166] and [167], which are fully asynchronous FL and asynchronous FL with periodic aggregation, as shown in Fig. 10. For the fully asynchronous FL in [167], in each time slot, the server receives a locally trained model $\boldsymbol{w}_{\text{new}}$ from an arbitrary device and updates the global model $\boldsymbol{w}_G$ by weight averaging, which is denoted as

$$\boldsymbol{w}_G^t = (1-\alpha)\boldsymbol{w}_G^{t-1} + \alpha \boldsymbol{w}_{\text{new}}, \tag{90}$$

where $\alpha \in (0,1)$ is a mixing hyperparameter to determine the contribution of the local model to the global model. Intuitively, a larger bias of the local model results in a higher error when updating the global model. The local model updating is still written as (6). Based on [167], the fully asynchronous FL was used in [168] for edge devices with non-IID data, so that the server does not need to wait for the devices with high communication delays.

However, fully asynchronous FL with sequential updating has the problem of high communication costs caused by frequent local model updating and transmission. To address this issue, an asynchronous FL with periodic aggregation and an adaptive asynchronous FL (AAFL) were proposed in [167] and [169], respectively.

For the asynchronous FL with periodic aggregation in [167], the edge server periodically collects local models to update the global model from devices that have completed local training. While other devices continue their local training without being interrupted or dropped. Particularly, after each device updates its local model by (6), it transmits a signal to the server indicating its completion of local model training. After each time duration T, the server schedules a subset of ready-to-update devices to upload their local models. The received local models are aggregated at the server by (7), and then the updated global model is distributed to these devices, and continue their local training based on the newly received global model.

The AAFL algorithm in [169] is an experience-driven algorithm based on deep reinforcement learning (DRL), which can adaptively determine the optimal fraction value α in each time slot. Given the completion time of the learning task, local model parameters, loss function, the difference between the current loss value and target loss value, bandwidth consumption, and remaining resource budget in each time slot, the DRL agent selects the value of α for model aggregation. Integrating AAFL with DRL reduces the training time and improves learning accuracy compared to fully asynchronous FL.

Based on the proposed synchronous and asynchronous FL, authors in [170] proposed a time-triggered FL (TT-Fed) over wireless networks, which was a generalized form of classic synchronous and asynchronous FL and achieved a good balance between training and communication efficiencies. The global model aggregation in TT-Fed is triggered in each fixed global model aggregation round duration $\triangle T$. Assuming that T is the time required for the slowest device to complete one single local updating round. Thus, all devices are partitioned into $M = \lceil \frac{T}{\triangle T} \rceil$ tiers ($\lceil . \rceil$ is the ceiling function), where the first tier is the fastest tier and the Mth tier is the slowest tier. As shown in Fig. 9, assuming that 4 devices are partitioned into 3 tiers according to the global model aggregation round duration partitioning. Device 1 and device 2 in the first tier need a single global model aggregation $\triangle T$ to complete their local updating, while device 4 in the third tier needs three $\triangle T$. Thus, the server has new updates from different tiers in each global model aggregation round. By using TT-Fed, it is possible for the global model to be broadcast to users in different tiers for communication overhead reduction.

Authors in [148], [150]–[152], [154], [156]–[158], [164], [166], [167], [169], [170] considered OFDMA and TDMA to transmit the local or global models. However, when a large number of devices uploading high-dimensional local models, the classic orthogonal-access schemes, such as OFDMA and TDMA, are not able to scale well with an increasing number of devices. To deal with this issue, over-the-air computation (OAC) has been proposed, it is a disruptive technology for fast data aggregation in wireless networks through exploiting the waveform superposition property of multi-access channels. In particular, the transmitted signal in the uplink transmission is superimposed over-the-air and their weighted sums, so-called the aggregated signal, are processed at the edge [171], [172].

B. Analog Approach of FL over Wireless Networks

In the digital approach, one challenge is to overcome the communication bottleneck, which is caused by many devices uploading high-dimensional models to a central server. A promising approach is to design new multiple access schemes, and a recently emerged approach, the so-called OAC, can provide the required scalability for FL over wireless networks.

OAC is a promising approach for fast wireless data aggregation via computing a nomographic function of distributed data from multiple devices in the uplink transmission, and it can accomplish the computation of target function by concurrent transmission, thereby significantly improving the communication efficiency compared to orthogonal transmission [173]. The uplink transmission includes local model weights or gradients, while the downlink information usually includes updated model weights or aggregated gradients.

The OAC-based approach for fast global model aggregation in the uplink transmission was proposed in [174] to explore the superposition property of a wireless multi-access channel via the joint device selection and beamforming design. In the tth time slot, the kth device transmits the signal $\mathbf{s}_k^t$ to the BS, and the received signal at the BS is expressed as

$$\mathbf{y} = \sum_{k=1}^{K} \mathbf{h}_k b_k \mathbf{s}_k + \mathbf{n}, \tag{91}$$

where $\mathbf{h}_k$ is the channel vector between the kth device and the BS, b_k is the transmitter scalar, and $\mathbf{n}$ is the noise vector.

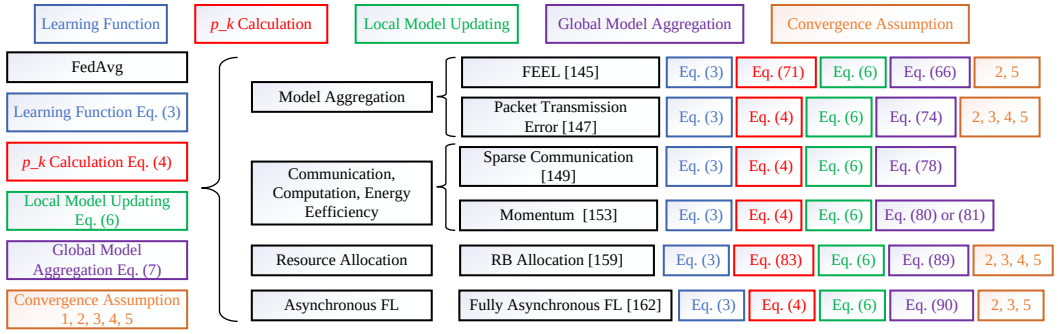


Fig. 11. Joint design of FL and wireless networks.

Through designing the receiver beamforming vector $\mathbf{m}$, the estimated global model at the BS is calculated as

$$\hat{\boldsymbol{w}_G} = \frac{1}{\sqrt{\eta}}\mathbf{m}^H\mathbf{y}, \tag{92}$$

where η is a normalizing factor. For the global model updating in the OAC, the difference between the estimated global model $\hat{\boldsymbol{w}}_G$ and the target function $\boldsymbol{w}_G$ should be minimized, and a mean-square-error (MSE) is used to quantify the performance, which is defined as

$$\text{MSE}(\hat{\boldsymbol{w}}_g, \boldsymbol{w}_g) = E(\|\hat{\boldsymbol{w}}_g - \boldsymbol{w}_g\|_2). \tag{93}$$

Motivated by [175], given the receiver beamforming vector $\mathbf{m}$, the MSE is minimized by using the zero-forcing transmitter

$$b_k = \sqrt{\eta}p_k\frac{(\mathbf{m}^H\mathbf{h}_k)^H}{\|\mathbf{m}^H\mathbf{h}_k\|_2}. \tag{94}$$

Based on [174], authors in [176] considered broadband analog aggregation (BAA) for OAC model aggregation to further maximize the number of scheduled devices under update-distortion constraints, which could reduce the communication latency. In BAA, the weights of local models are first modulated into symbols. Then, the symbol sequence is divided into blocks, and each block is transmitted in a single OFDM symbol over one frequency sub-channel. Sub-channels are inverted by power control, so that weights transmitted by different devices are received with identical amplitudes, achieving amplitude alignment at the receiver as required for BAA.

Different from BAA, only transmitting the sign of gradients in OAC also enables a large number of devices to participate in model aggregation and achieve convergence, which has been exploited in [177]–[180]. In the downlink transmission, broadcasting updated model weights guarantees that the devices compute the gradients based on the same model weights. On the other hand, broadcasting aggregated gradients in multi-cell OAC promotes the personalization of model weights for devices located at the cell edge [181].

According to [177]–[181], various techniques have been developed to investigate the performance of OAC, which can find its application in distributed sensing and autonomous control [182]. In distributed sensing, OAC is able to achieve efficient distributed sensing by conducting simultaneous transmission among all sensors, and the desired function can be directly computed over the air. In autonomous control, a group of agents desire to perform some actions to accomplish an overall cooperative task by interacting with each other. As a result, each agent needs to iteratively collect information from others for updating its own state until convergence, which generally consists of two phases within each iteration, i.e., the communication phase for information exchanges and the computation phase for state updates. By integrating these two phases, efficient network-wide consensus can be achieved by OAC in a distributed manner. However, authors in [177]–[181] generally assumed some ideal communication conditions and ignore practical implementation issues.

Since OAC focuses on uplink transmission, time-division duplexing (TDD) is commonly considered in the existing research works to achieve local CSI estimation at each device based on the pilot signal broadcast by the edge server. In TDD systems, the uplink and downlink transmissions are carried out in the same frequency channel but in different time slots. Therefore, the edge server first broadcasts pilot signals to devices for estimating the local CSI, and then obtains the global CSI based on the feedback from devices by assuming the channel reciprocity. However, the communication overhead is linearly scaling with the number of devices for CSI feedback, which may lead high transmission latency in ultra-dense networks. To address this issue, effective channel feedback approaches should be developed to realize OAC while avoiding massive overhead for CSI gathering. In [183], random orthogonalization was proposed for FL in massive MIMO systems, which significantly reduced the channel estimation overhead while achieving OAC model aggregation without requiring transmitter side CSI.

Furthermore, in practical systems, the transceivers can only obtain imperfect CSI because of multiple reasons, such as inaccurate channel estimation and finite-rate feed back. Therefore, simply implementing the system design based on the assumption of perfect CSI may lead to poor learning performance in practical wireless networks with imperfect CSI. To address this issue, robust design of OAC is required to achieve a reliable functional computation under imperfect CSI. In [184], a maximum-likelihood estimation design for misaligned AirComp with residual channel-gain variation and symbol-timing asynchrony among devices was proposed. To further address the error propagation and noise enhancement problems

at the maximum likelihood estimator, a whitened matched filtering and sampling (WMFS) scheme was deployed.

In addition, most of the current design for OAC relies on the instantaneous CSI at the edge server or devices. However, the cumbersome CSI acquisition introduces extra transmission latency and communication overhead, which motivates the CSI-free design, so-called blind design for OAC. To mitigate the possible destructive signal superposition due to the phase difference when CSI is not available, authors in [185], [186] considered the one-bit quantization to realize blind OAC, where the receiver obtains the sign of aggregated signals based on majority vote by detecting the energy accumulated on different OFDM subcarriers [185] and superposed pulse-position modulation symbols [186], which avoided the reliance on the CSI and relaxed the requirement of synchronization. In addition, the balanced number system was considered in [187] to enable continuous-valued computations in the digital OAC system without the need of CSI acquisition.

It should be noted that FL with OAC inherits the common problems in FL literature such as convergence under different data distributions, device heterogeneity, stragglers, data privacy, and various security issues. Therefore, these application-specific challenges need to be re-evaluated for a given OAC scheme.

C. Convergence Analysis of FL over Wireless Networks

The derivation of convergence analysis of FL is still based on the fundamental assumptions 1 - 4 in Section III. Different from the convergence analysis of FL methodologies, researchers usually derive the upper bound of $\mathbb{E}\{f(\boldsymbol{w})-f(\boldsymbol{w}^*)\}$ or $\mathbb{E}\{\frac{1}{T}\sum_{t=0}^{T-1}\|\boldsymbol{g}^t\|\}$ in FL over wireless networks. We will present several examples as follows.

In [152], except from assumptions 1 - 4, authors added another assumption in the following.

Assumption 5. We assume that $f_n(\boldsymbol{w})$ is strongly convex with positive parameter μ, such that

$$f(\boldsymbol{w}^{t+1}) \geq f(\boldsymbol{w}^t)+(\boldsymbol{w}^{t+1}-\boldsymbol{w}^t)^T\nabla f(\boldsymbol{w}^t)+\frac{\mu}{2}\|\boldsymbol{w}^{t+1}-\boldsymbol{w}^t\|^2. \tag{95}$$

The upper bound of $\mathbb{E}\{f(\boldsymbol{w})-f(\boldsymbol{w}^*)\}$ is derived as

$$\mathbb{E}\{f(\boldsymbol{w}^{t+1})-f(\boldsymbol{w}^*)\} \leq B^t\mathbb{E}\{f(\boldsymbol{w}^0)-f(\boldsymbol{w}^*)\} + \frac{2G}{LD}\sum_{k=1}^{K}D_k(1-a_k+a_kq_k(r_k,P_k))\frac{1-B^t}{1-B}, \tag{96}$$

where D is the total number of training data, a_k, q_k, r_k, and P_k have already been defined in (75) - (78), and B is denoted as

$$B = 1-\frac{\mu}{L}+\frac{4\mu G}{LD}\sum_{k=1}^{K}D_k(1-a_k+a_kq_k(r_k.P_k)). \tag{97}$$

It is observed that there is a gap $\frac{2G}{LD}\sum_{k=1}^{K}D_k(1-a_k+a_kq_k(r_k,P_k))\frac{1-B^t}{1-B}$ between $\mathbb{E}\{f(\boldsymbol{w}^{t+1})\}$ and $\mathbb{E}\{f(\boldsymbol{w}^*)\}$, which is caused by the packet errors and device selection policy. The gap decreases when the packet error rate decreases or the number of devices participating the model aggregation increases. Also, the value of B decreases with decreasing packet error rate, which means that the convergence rate of FL improves.

In [177], based on assumptions 2 - 4, the convergence rate of OAC in FL over AWGN channels is given by

$$\mathbb{E}\{\frac{1}{T}\sum_{t=0}^{T-1}\|\boldsymbol{g}^t\|\} \leq \frac{a_{\text{AWGN}}}{\sqrt{T}}(\sqrt{L}(f(\boldsymbol{w}^0)-f(\boldsymbol{w}^*)+\frac{\gamma}{2})+ \frac{2\gamma}{\sqrt{K}}\hat{\sigma}+b_{\text{AWGN}}), \tag{98}$$

where the scaling factor a_{AWGN} and the bias term b_{AWGN} are denoted as

$$a_{\text{AWGN}} = \frac{1}{1-\frac{1}{K\sqrt{\hat{\rho}}}},\ b_{\text{AWGN}}\frac{2\gamma\hat{\sigma}}{K\sqrt{\hat{\rho}}}. \tag{99}$$

In (98) and (99), $\gamma > 0$ is a positive constant, and $\hat{\rho}$ denotes the receive SNR. (98) means that the existence of channel noise slows down the convergence rate by adding a scaling factor a_{AWGN} and a positive bias term b_{AWGN} to the upper bound on the time-averaged gradient norm. Therefore, more communication rounds are required for convergence. However, the negative effect of channel noise vanishes at a scaling rate of $\frac{1}{K}$ with increasing number of devices participating in the model aggregation.

D. Summary and Lessons Learned

In this section, we have reviewed two main research areas of FL over wireless networks. We summarize the approaches along with references. From this review, we gather the following lessons learned:

- Wireless factors, such as transmission power, wireless channel, and spectrum resource allocation, affect the convergence and learning accuracy of FL over wireless networks. The evolution of FedAvg in wireless networks is shown in Fig. 11. According to Fig. 11, through jointly or separately optimizing equations (3), (4), (6), and (7), FL can be more appropriate for wireless networks. In addition, theoretical convergence analysis is usually derived based on five fundamental assumptions.
- Synchronous FL systems are susceptible to the straggler effect. As a result, asynchronous FL has been proposed to solve the problem. Asynchronous FL allows devices to participate in model training even a training round is in progress. This is more reflective of practical FL scenarios and is an important contribution towards guaranteeing the scalability of FL. However, synchronous FL is still the most common approach used because of its convergence guarantees. Therefore, the convergence analysis in asynchronous FL settings needs to be investigated.
- For the research works we have discussed in this section, most works assume that the wireless transmission between servers and devices is successful and error-free. However, in wireless networks, because of network congestion, interference, and bit error, the model transmission between servers and devices may fail or contain errors, which further affects learning performance. Therefore, robust FL algorithms over wireless networks should be investigated to address the issue.

- In this section, we observe that FL usually focuses on model training for a single ML task across multiple devices, namely, the well-trained model weights cannot generalize to multiple tasks. To address this issue, meta learning is discussed in the following sections.

V. Meta Learning Methodologies

Meta learning is most commonly understood as learning to learn, which refers to a learning algorithm that can generalize across different tasks. Thus, it has an advantage over traditional data-driven ML algorithms that can only work on a single task with the well-trained weights $\boldsymbol{w}$ [68], [70], [71]. Therefore, meta learning is able to help FL adapt to multiple tasks. Specifically, in meta learning, we evaluate the performance of weights $\boldsymbol{w}$ over a distribution of tasks $p(\mathcal{T})$. We loosely assume a task consisting of a dataset and loss function $\mathcal{T} = \{\mathcal{D}, \mathcal{L}\}$. The objective of meta learning is to minimize the expectation of loss function over all tasks, which is given by

$$\min_{\boldsymbol{w}} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \mathcal{L}(\mathcal{D}; \boldsymbol{w}), \tag{100}$$

where $\mathcal{L}(\mathcal{D}; \boldsymbol{w})$ measures the performance of a model trained using weights $\boldsymbol{w}$ on dataset $\mathcal{D}$. To solve the problem in (100), a set of M source tasks are sampled from $p(\mathcal{T})$ and used in the meta-training stage as $\mathcal{D}_{source} = \{(\mathcal{D}_{source}^{train}, \mathcal{D}_{source}^{test})^i\}_{i=1}^{M}$, where each task has both training and testing data. Also, the source training and testing datasets are usually called support and query sets, respectively. The meta-training step of "learning how to learn" is written as

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}|\mathcal{D}_{source}). \tag{101}$$

Then, a set of Q target tasks used in the meta-testing stage is denoted as $\mathcal{D}_{target} = \{(\mathcal{D}_{target}^{train}, \mathcal{D}_{target}^{test})^i\}_{i=1}^{Q}$, where each task has both training and testing datasets. In the meta-testing stage, we use the learned weights $\boldsymbol{w}$ to train the weights of each new target task i, which is denoted as

$$\boldsymbol{\theta}^{*(i)} = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\boldsymbol{w}^*, \mathcal{D}_{target}^{train(i)}). \tag{102}$$

According to (102), we can obtain that learning on the training set of a target task i benefits from meta-knowledge $\boldsymbol{w}^*$, and evaluate the accuracy of meta-learner by the performance of $\boldsymbol{\theta}^{*(i)}$ on the testing dataset of each target task $\mathcal{D}_{target}^{test(i)}$. Meta-learning algorithms can be categorized into three main directions: (1) metric-based, (2) model-based, and (3) gradient-based optimization methods, which are introduced in detail in the following subsections. The introduced meta learning methodologies are fundamental meta learning algorithms from the CS community without considering any wireless factors.

A. Metric-based Meta Learning

Metric-based methods learn the meta knowledge $\boldsymbol{w}$ through a feature space that is used for various new tasks. The feature space is integrated with the weights $\boldsymbol{\theta}$ of the neural networks. Then, new tasks are learned by comparing new inputs with example inputs in the meta-learned feature space. The higher the similarity between the new input and the example input, the more likely that the new input has the same label as the example input. Thus, metric-based meta-learning aims to learn a similarity kernel that takes two inputs, and outputs their similarity score. Larger similarity scores present larger similarities. In this subsection, we introduce four key metric-based meta-learning methods, including siamese networks, matching networks, prototypical networks, and relation networks. The relationship among these four methods is presented in Fig. 12.

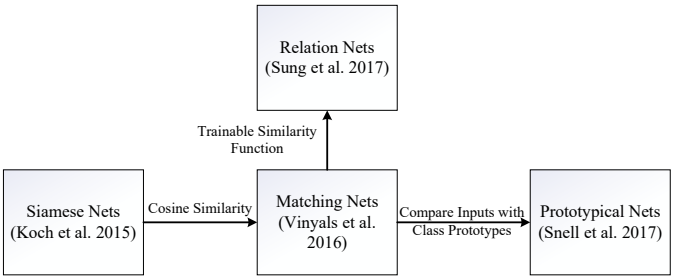


Fig. 12. Development of metric-based meta learning.

1) Siamese Networks: Authors in [188] used a siamese network to compare the distance between data samples. A siamese network consists of two neural networks that share the same weights $\boldsymbol{\theta}$. It takes two inputs $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, and computes two hidden states $f_{\boldsymbol{\theta}}(\boldsymbol{x}_1)$ and $f_{\boldsymbol{\theta}}(\boldsymbol{x}_2)$. Then, these two hidden states are input into a distance layer to calculate a distance vector, which is given by

$$\boldsymbol{d} = |f_{\boldsymbol{\theta}}(\boldsymbol{x}_1) - f_{\boldsymbol{\theta}}(\boldsymbol{x}_2)|. \tag{103}$$

According to the distance vector $\boldsymbol{d}$, we can obtain whether two inputs $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ belong to the same class. The siamese network is a simple approach in metric-based meta-learning, and can only be deployed to supervised learning scenarios.

2) Matching Networks: Based on the distance comparison idea in siamese networks, authors in [189] proposed a matching network to learn the similarity between support sets and new inputs from query sets. The matching networks use a weighted combination of all example labels in the support set and an attention kernel to compute the similarity of inputs $\boldsymbol{x}_i$ and new input $\boldsymbol{x}$. The attention kernel uses the cosine distance [190] to calculate the similarity of the input representations, rather than using the distance vector in (103) to quantify the similarity of two inputs. The matching network is still a simple approach in metric-based meta-learning, and is not applicable outside of the supervised learning scenarios. Furthermore, it suffers from performance degradation when label distributions are biased.

3) Prototypical Networks: Similar to matching networks, prototypical networks proposed in [191] also used samples in the support set. However, rather than calculating the similarity between samples in the support set and new inputs, prototypical networks map inputs to a dimensional vector space such that inputs of a given output class are close together. Since the number of class prototypes is smaller than that of samples in the support set, the amount of comparisons decreases, which further reduces computational costs. However, prototypical networks can only be used in supervised learning scenarios.

4) Relation Networks: Different from the pre-defined similarity metric in siamese and matching networks, relation networks (RN) proposed in [192] used a trainable similarity metric. RN consists of two modules, which are an embedding module responsible for embedding inputs, and a relation module computing similarity scores between new inputs $\boldsymbol{x}$ and example inputs $\boldsymbol{x}_i$ from support sets. Then, a classification decision is made by selecting the class of the example input which outputs the highest similarity score. RN uses the Mean-Squared Error (MSE) as a similarity score, and the MSE is then propagated backward through the entire network to update the weights in embedding and relation modules. Because of the trainable similarity metric, the accuracy performance of RN is better than that of siamese and matching networks with a fixed similarity metric.

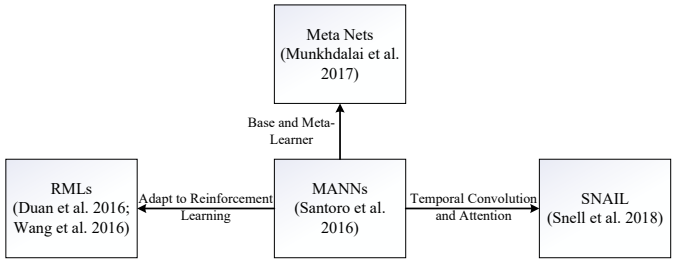


Fig. 13. Development of model-based meta learning.

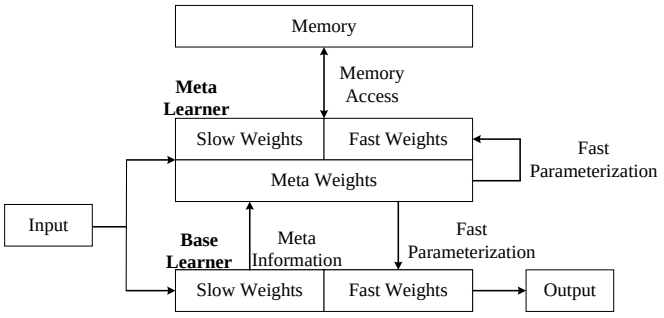


Fig. 14. Architecture of meta network.

B. Model-based Meta Learning

In contrast to the metric-based approaches deploying fixed neural networks at the testing phase, model-based meta-learning algorithms depend on the internal state of each task. Specifically, model-based approaches process the support set in a sequential fashion. In each time slot, the internal state captures relevant task-specific information with the given inputs, which can be used to make predictions for new inputs. Meanwhile, task information from previous inputs should be remembered, so that model-based methods have a memory component. In this subsection, we introduce four key model-based meta-learning methods, including memory-augmented neural networks (MANNs), meta networks (MetaNets), recurrent meta-learners (RMLs), and simple neural attentive meta-learner (SNAIL). The relationship of these four networks are shown in Fig. 13.

1) Memory-augmented Neural Networks: MANNs were proposed in [193] to allow for quick task-specific adaptation with the help of a neural turing machine (NTM) [194] and an external memory. The learning procedure of MANNs is that the data of a task is processed as a sequence. First, the support set is input to MANN. Then, the query set is evaluated. The interaction between NTM and external memory is that NTM gradually accumulates meta knowledge across tasks, and the external memory helps to store the obtained knowledge. Given new inputs, NTM leverages the previously obtained meta knowledge stored in the external memory to make predictions. MANNs integrate the external memory and a neural network to achieve meta learning. Different from metric-based meta-learning, MANN can be used for both classification and regression problems. However, it has higher architectural complexity.

2) Meta Networks: Similar to MANN, MetaNets proposed in [195] also leveraged an external memory to store the meta knowledge. However, different from MANN, MetaNets are divided into two distinct subsystems, which are base-learner and meta-learner, as shown in Fig. 14. The base-learner is used to perform tasks, and provide meta knowledge for the meta-learner. Then, the meta-learner calculates fast task-specific weights for itself and the base-learner. The training of MetaNet consists of three main procedures: (1) Acquisition of meta knowledge; (2) Generation of fast weights; (3) Optimization of slow weights. MetaNets depend on base-learner and meta-learner for each task. Although it can be used for both supervised and reinforcement learning scenarios, the learning architecture is quite complex and leads to a high burden on memory usage and computation time.

3) Recurrent Meta-learner: RMLs proposed in [196] and [197] were meta-learners based on recurrent neural networks (RNNs), and were specifically proposed for reinforcement learning scenarios. The internal learning architecture of the selected RNN allows for fast adaptation to new tasks. Similar to MANN, RML still uses memory to store the meta knowledge and the task data is sequentially input into the learning model. However, RMLs have simple learning architectures, mainly perform well on simple reinforcement learning tasks, and cannot be adapt to complex learning scenarios.

4) Simple Neural Attentive Meta-learner: Similar to MANN, SNAIL proposed in [198] still processes task data in sequence. However, rather than using external memory, SNAIL deploys a special model architecture to serve as memory. The special model consists of 1D temporal convolutions [199] and a soft attention mechanism [200]. The 1D convolutions are used for memory access, and the attention mechanism allows SNAIL to pinpoint specific experiences. Furthermore, SNAIL contains three building blocks, which are DenseBlock, TCBlock, and AttentionBlock. The DenseBlock deploys a single 1D convolution to the input and connects to the result, the TCBlock consists of a series of DenseBlocks, and the AttentionBlock learns the important parts of prior experience. A key advantage of SNAIL is that it can be used for both supervised and reinforcement learning scenarios, and it achieves better learning accuracy performance than that of the other three model-based meta-learning algorithms.

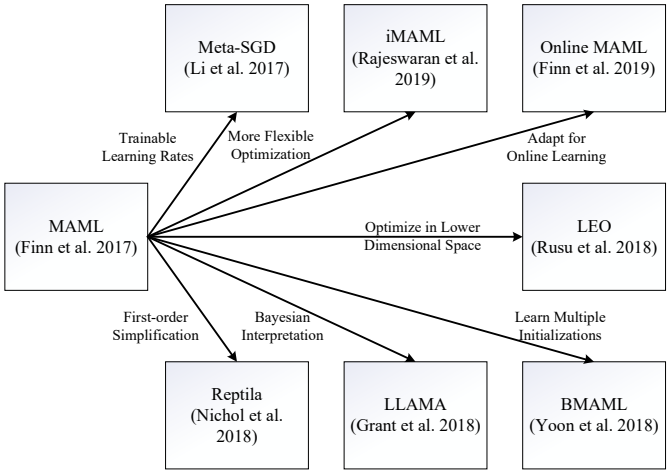


Fig. 15. Development of MAML.

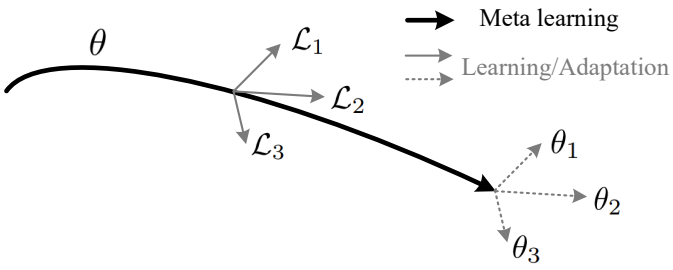


Fig. 16. Diagram of MAML.

C. Gradient-based Meta Learning

Different from the metric-based and model-based meta learning approaches, gradient-based meta-learning is mainly trained with an interleaved training procedure, including inner loop of task-specific adaptation and outer loop of meta initialization training [201], [202]. The traditional gradient-based meta learning is model-agnostic meta learning (MAML). Based on MAML, several other advanced meta learning models have been proposed, which are meta-SGD, reptile, Bayesian MAML (BMAML), Laplace approximation for meta adaptation (LLAMA), latent embedding optimization (LEO), MAML with Implicit Gradients (iMAML), and online MAML, as shown in Fig. 15.

1) MAML: MAML is a model and task-agnostic algorithm for meta-learning that trains model weights with a small number of gradient steps and leads to fast learning on a new task [203]. Thus, MAML has two advantages: 1) it can be fine-tuned, which means that it quickly adapts to new tasks, and 2) it requires fewer training samples [204]. The learning trend of MAML is shown in Fig. 16. From the learned initialization weights $\boldsymbol{\theta}$, MAML can quickly move to the optimal set of weights $\boldsymbol{\theta}_i^*$ for the task $\mathcal{T}_i$ $(i = 1, 2, 3)$.

The meta learning model is represented by a function $f_{\boldsymbol{\theta}}$ with weights $\boldsymbol{\theta}$. When it adapts to a new task $\mathcal{T}_i$, the model weights $\boldsymbol{\theta}$ become $\boldsymbol{\theta}_i^{'}$. The updated weights $\boldsymbol{\theta}_i^{'}$ is computed by using gradient descent on task $\mathcal{T}_i$, which is denoted as

$$\boldsymbol{\theta}_i^{'} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}}). \tag{104}$$

Algorithm 2 Model-Agnostic Meta-Learning

```
p(T):distribution over tasks.
Initialize step size hyperparameters α and β, randomly initialize learning weights θ.
while not done do
    Sample batch of tasks T_i ~ P(T).
    for all T_i do
        Evaluate ∇_θ L_{T_i}(f_θ) with respect to K data samples.
        Calculate adapted weights with gradient descent: θ_i' = θ − α∇_θ L_{T_i}(f_θ).
    end for
    Update θ = θ − β∇_θ Σ_{T_i∼p(T)} L_{T_i}(f_{θ_i'}).
end while
```

The step size $\alpha \in (0, 1)$ can be fixed as a hyperparameter or dynamically meta-learned. The learning model weights are trained by optimizing the performance of $f_{\boldsymbol{\theta}_i^{'}}$ with respect to $\boldsymbol{\theta}$ across the tasks sampled from $p(\mathcal{T})$. More concretely, the meta-objective is to minimize the loss function of all tasks with the learned initialization weights $\boldsymbol{\theta}$, which is presented as

$$\min_{\boldsymbol{\theta}} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}_i^{'}}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}})}), \tag{105}$$

where $\mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}_i^{'}})$ is the loss function of the ith task $\mathcal{T}_i$ with its model weights $\boldsymbol{\theta}_i^{'}$, and $\boldsymbol{\theta}_i^{'}$ is updated by (104). According to (104) and (105), the optimization of meta learning is performed over the initialized model weights $\boldsymbol{\theta}$, and the objective is achieved by the updated model weights $\boldsymbol{\theta}^{'}$. The initialized model weights $\boldsymbol{\theta}$ are updated as

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}_i^{'}}), \tag{106}$$

where β is the meta step size, and $\nabla_{\boldsymbol{\theta}} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}_i^{'}})$ is calculated as

$$\nabla_{\boldsymbol{\theta}} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}_i^{'}}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} (\mathbf{I} - \alpha \nabla^2 f_i(\boldsymbol{\theta})) \nabla f_i(\boldsymbol{\theta} - \alpha \nabla f_i(\boldsymbol{\theta})). \tag{107}$$

In (107), $\mathbf{I}$ is an identity matrix. The detailed procedures of MAML are introduced in **Algorithm** 2. Based on the same assumptions of FL methodologies in Section III, the upper bound of $\mathbb{E}\|\nabla \mathcal{L}(f_{\boldsymbol{\theta}})\|$ is derived as

$$\mathbb{E}\|\nabla \mathcal{L}(f_{\boldsymbol{\theta}})\| \leq \mathcal{O}\left(\sqrt{\frac{\hat{\sigma}^2}{B} + \frac{\hat{\sigma}^2}{BD_{\mathrm{o}}} + \frac{\hat{\sigma}^2}{D_{\mathrm{in}}}}\right) + \epsilon, \tag{108}$$

where B is the number of tasks, D_{o} is the size of datasets of outer loop, and D_{in} is the size of datasets of inner loop, $\hat{\sigma}$ is defined in assumption 4, and $0 < \epsilon < 1$. (108) means that if the batch sizes B, D_{o}, and D_{in} are selected properly, for any $\epsilon > 0$, MAML is able to converge after limited number of iterations.

However, calculating Hessian vector $\nabla^2 f_i(\boldsymbol{\theta})$ in (107) increases computation complexity, which can result in high computation latency [205]. To address this issue, first-order MAML (FO-MAML) was proposed in [206], where the authors directly ignored $\nabla^2 f_i(\boldsymbol{\theta}))$. Surprisingly, the convergence

performance of this method is nearly the same as that with full second derivatives, suggesting that most of the performance improvement in MAML comes from the gradients of the objective at the first-order update, rather than the second updates from differentiating through the gradient update. Previous research works observed that ReLU neural networks were almost locally linear, which suggested that the second derivatives may be close to zero in most cases, partially explaining the performance improvement of the first-order approximation [207]. In addition, the first-order approximation can achieve 33% speed-up in terms of network computation.

To further reduce the computation time, authors in [208] introduced Hessian-free MAML (HF-MAML), which did not require computation of the Hessian vectors, and its computation complexity was the same as that of FO-MAML, but it achieved better convergence rate than FO-MAML. It is because Hessian-free is a method to avoid the vanishing gradient problem while using backpropagation in DNNs [209]–[212]. The idea behind HF-MAML is that for any function ϕ, the product of its Hessian $\nabla^2\phi(\boldsymbol{\theta})$ and any vector $\mathbf{v}$ can be approximated as

$$\nabla^2\phi(\boldsymbol{\theta})\mathbf{v} \approx \left[\frac{\nabla\phi(\boldsymbol{\theta}+\delta\mathbf{v}) - \nabla\phi(\boldsymbol{\theta}-\delta\mathbf{v})}{2\delta}\right], \tag{109}$$

with an error of at most $\rho\delta\|\mathbf{v}\|^2$, where ρ is the parameter for Lipschitz continuity of the Hessian of $\phi(\boldsymbol{\theta})$. By integrating (109) into (107), we can obtain that

$$\phi(\boldsymbol{\theta}) = f_i(\boldsymbol{\theta}), \mathbf{v} = \nabla f_i(\boldsymbol{\theta} - \alpha\nabla f_i(\boldsymbol{\theta})). \tag{110}$$

Thus, $\nabla^2 f_i(\boldsymbol{\theta})\nabla f_i(\boldsymbol{\theta} - \alpha\nabla f_i(\boldsymbol{\theta}))$ in (107) is calculated by (110), which decreases the computation complexity.

2) Meta-SGD: Meta-SGD was proposed in [213], which is an easily trainable meta-learner that could initialize and adapt any learner in just one step, for both supervised learning and reinforcement learning. Different from MAML, Meta-SGD can achieve a much higher accuracy not only by the learner initialization, but also by the learner update direction and learning rate, all in a single meta learning process. The main difference is the way it updates $\boldsymbol{\theta}_i^{'}$, different from MAML that updates $\boldsymbol{\theta}_i^{'}$ based on (104), Meta-SGD updates $\boldsymbol{\theta}_i^{'}$ using

$$\boldsymbol{\theta}_i^{'} = \boldsymbol{\theta} - \boldsymbol{\alpha} \circ \nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}}), \tag{111}$$

where $\boldsymbol{\alpha}$ is a vector of the same size as $\boldsymbol{\theta}$ that determines both the update direction and learning rate, and $\circ$ denotes the element-wise product. The adaptation term $\boldsymbol{\alpha} \circ \nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}})$ is a vector whose direction denotes the update direction. Note that learning rate $\boldsymbol{\alpha}$ in (111) is a vector rather than a scalar in (104), and (111) allows for a higher flexibility in the sense that each weight has its own learning rate. Compared with MAML, Meta-SGD is easier to implement and can learn more efficiently due to that both of its update direction and learning rate can be optimized.

3) Reptile: Like MAML, Reptile learns a weight initialization that can be fine-tuned quickly on a new task. However, the way in which Reptile tries to obtain the optimal weights is quite different from MAML. Given the initialized model weight $\boldsymbol{\theta}$, it works by repeatedly sampling only one task in each time slot, training on it, and moving the initialized weights towards the trained weights on that task [214]. As shown in Fig. 17, the initialized weights $\boldsymbol{\theta}$ are moving towards the optimal weights for tasks 1 or 2. Because in each time slot, only one task is selected to train the learning weights, the initialized weights $\boldsymbol{\theta}$ oscillate between tasks 1 and 2. For example, if selecting task 1 to train $\boldsymbol{\theta}$, $\boldsymbol{\theta}$ is moving towards $\boldsymbol{\theta}_1^{'}$ and is updated as $\boldsymbol{\theta}_1$ after several iterations. Then, if stop using task 1 and selecting task 2 to train $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_1$ is moving towards $\boldsymbol{\theta}_2^{'}$ and is updated as $\boldsymbol{\theta}_2$ after several iterations. Therefore, Reptile is a very simple meta-learning algorithm, and does not require updating model weights through the optimization process like MAML, making it more suitable for optimization problems where a limited number of update steps are required, and saving time and memory costs. In Reptile, $\boldsymbol{\theta}_i^{'}$ for the ith task is updated using (111). However, for the initialization weights $\boldsymbol{\theta}$, it moves toward to the trained weights, which is updated as

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \beta(\boldsymbol{\theta}_i^{'} - \boldsymbol{\theta}). \tag{112}$$

In (112), $\boldsymbol{\theta}_i^{'} - \boldsymbol{\theta}$ is the distance between initialization weights $\boldsymbol{\theta}$ and learning weights for the ith task $\boldsymbol{\theta}_i^{'}$. The Reptile algorithm is shown in **Algorithm** 3. Although the Reptile is an extremely simple meta learning technique, the convergence and accuracy performance may be a bit worse than that of MAML because of its simple learning procedure.

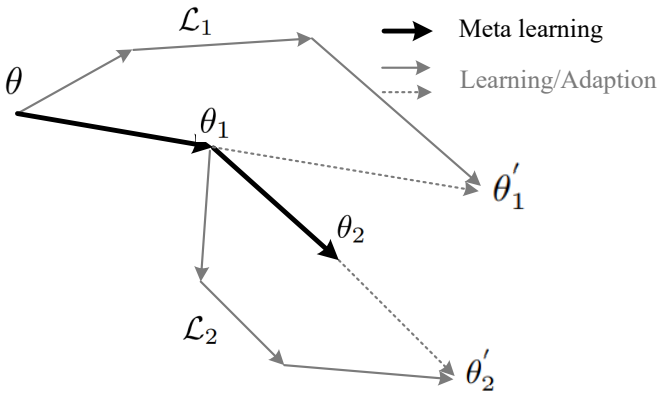


Fig. 17. Diagram of Reptile.

Algorithm 3 Reptile Meta-Learning

1: $p(\mathcal{T})$:distribution over tasks
2: Initialize step size hyperparameters β, randomly initialize learning weights $\boldsymbol{\theta}$
3: **for** i = 1,2,... **do**
4: Sample batch of tasks $\mathcal{T}_i \sim P(\mathcal{T})$.
5: Calculate $\boldsymbol{\theta}_i^{'}$ via (111).
6: Calculate initialization weights $\boldsymbol{\theta}$ via (112).
7: **end for**

4) BMAML: Unlike MAML that learns a distribution over potential solutions, Bayesian MAML (BMAML) in [215] learns M possible weights $\boldsymbol{\Theta} = \{\boldsymbol{\theta}\}_{i=1}^{M}$ and jointly optimizes them in parallel. To update these weights, authors in [215] deployed Stein Variational Gradient Descent (SVGD) [216]. SVGD is a non-parametric variational inference method, which leverages the advantages of Markov chain Monte Carlo

(MCMC) [217] and variational inference. Also, it converges faster than MCMC because its update rule is deterministic. Specifically, SVGD maintains M instances of model weights, called particles. In the tth time slot, each model weight vector $\boldsymbol{\theta}_t \in \boldsymbol{\Theta}$ is updated using

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\phi(\boldsymbol{\theta}_t), \tag{113}$$

where α is learning rate, and $\phi(\boldsymbol{\theta}_t)$ is given as

$$\phi(\boldsymbol{\theta}_t)=\frac{1}{M}\sum_{m=1}^{M}\Big[k(\boldsymbol{\theta}_t^m,\boldsymbol{\theta}_t)\nabla_{\boldsymbol{\theta}_t^m}\log p(\boldsymbol{\theta}_t^m)+\nabla_{\boldsymbol{\theta}_t^m}k(\boldsymbol{\theta}_t^m,\boldsymbol{\theta}_t)\Big]. \tag{114}$$

Here, $k(\boldsymbol{\theta}_t^m,\boldsymbol{\theta}_t)$ in (114) is a similarity kernel between $\boldsymbol{\theta}_t^m$ and $\boldsymbol{\theta}_t$. In (114), the update of one particle relies on the other gradients of particles, $k(\boldsymbol{\theta}_t^m,\boldsymbol{\theta}_t)\nabla_{\boldsymbol{\theta}_t^m}\log p(\boldsymbol{\theta}_t^m)$ moves the particle in the direction of gradients of other particles based on particle similarity, and $\nabla_{\boldsymbol{\theta}_t^m}k(\boldsymbol{\theta}_t^m,\boldsymbol{\theta}_t)$ enforces repulsive force between particles so that they do not collide to a same point. Then, these particles are used to approximate the probability distribution of labels in testing datasets, which is denoted as

$$p(y_j^{\text{test}}|\boldsymbol{\theta}_j^{'}) = \frac{1}{M}\sum_{m=1}^{M} p(y_j^{\text{test}}|\boldsymbol{\theta}_{\mathcal{T}_j}^m), \tag{115}$$

where $\boldsymbol{\theta}_{\mathcal{T}_j}^m$ is the mth particle calculated by training the support dataset (training dataset) $\mathcal{D}_{\mathcal{T}_j}^S$ of the task $\mathcal{T}_j$, and $p(y_j^{\text{test}}|\boldsymbol{\theta}_{\mathcal{T}_j}^m)$ is the data likelihood of the task $\mathcal{T}_j$.

To train BMAML, authors in [215] proposed a novel meta loss, called Chaser Loss. This loss aims to minimize the distance between the approximated parameter distribution achieved from the support set $p_{\mathcal{T}_j}^n(\boldsymbol{\theta}_{\mathcal{T}_j}|\mathcal{D}^S,\boldsymbol{\Theta}_0)$ and true distribution $p_{\mathcal{T}_j}^{n+s}(\boldsymbol{\theta}_{\mathcal{T}_j}|\mathcal{D}^S\cup\mathcal{D}^Q)$. Here, n is the number of SVGD steps, and $\boldsymbol{\Theta}_0$ is the set of initial particles. Because the true distribution is unknown, we need to approximate it by running SVGD for s additional steps to obtain $\boldsymbol{\Theta}_{\mathcal{T}_j}^{n+s}$, where s additional steps are executed on both the support and query sets. The proposed meta-loss is written as

$$\mathcal{L}_{\text{BMAML}}(\boldsymbol{\Theta}_0) = \sum_{\mathcal{T}_j\in B}\sum_{m=1}^{M}\|\boldsymbol{\theta}_{\mathcal{T}_j}^{n,m} - \boldsymbol{\theta}_{\mathcal{T}_j}^{n+s,m}\|_2^2, \tag{116}$$

where B is the number of sampled tasks. The BMAML algorithm is shown in **Algorithm 4**, where $d(\boldsymbol{\Theta}_{\mathcal{T}_j}^n(\boldsymbol{\Theta}_0),\boldsymbol{\Theta}_{\mathcal{T}_j}^{n+s}(\boldsymbol{\Theta}_0))$ is the dissimilarity between two distributions $\boldsymbol{\Theta}_{\mathcal{T}_j}^n(\boldsymbol{\Theta}_0)$ and $\boldsymbol{\Theta}_{\mathcal{T}_j}^{n+s}(\boldsymbol{\Theta}_0)$.

BMAML is a robust optimization-based meta learning that can generate M potential solutions for a task. However, it has to store M parameter sets in memory over time, which results in substantial memory costs.

5) LLAMA: Authors in [218] reformulated MAML as a method for probabilistic inference in a hierarchical Bayesian model. Through integrating MAML into a probabilistic framework, a probability distribution over task-specific weights $\boldsymbol{\theta}_j^{'}$ is learned, and multiple potential solutions can be obtained for a task. This extended MAML is called Laplace approximation for meta adaptation (LLAMA). To minimize the error on the query set $\mathcal{D}_{\mathcal{T}_j}^Q$, the model must output large probability scores for true classes. The log-likelihood loss function is denoted as

$$\mathcal{L}_{\mathcal{D}_{\mathcal{T}_j}^Q}(\boldsymbol{\theta}_j^{'}) = -\sum_{(\mathbf{x}_i,y_i)\in\mathcal{D}_{\mathcal{T}_j}^Q}\log P(y_i|\mathbf{x}_i,\boldsymbol{\theta}_j^{'}). \tag{117}$$

To predict the correct label y_i, authors in [218] deployed ML-Laplace to compute task-specific weights $\boldsymbol{\theta}_j^{'}$ updated from the initialization weights $\boldsymbol{\theta}$, and estimated the negative log-likelihood. The detailed LLAMA algorithm is shown in **Algorithms 5** and **6**.

LLMAMA extends MAML in a probabilistic style, which means that there are multiple potential solutions for a single task. However, it can only be deployed for supervised learning with high computational costs, and the Laplace approximation in ML-LAPLACE may be inaccurate, which further decreases the accuracy.

6) LEO: MAML operates in a high-dimensional parameter space using gradient information from only a few data samples from the support set, which can result in poor generalization. To deal with this issue, authors in [220] proposed a latent embedding optimization (LEO) to learn a lower dimensional latent embedding space, which indirectly updates a set of initialized weights $\boldsymbol{\theta}$. The detailed procedures of LEO are shown in Fig. 18. Given a task $\mathcal{T}_i$, the data samples from the support set pass through a stochastic encoder to produce $(Nk)^2$ pairs of hidden codes, where N is the number of classes in the support set, and k is the number of data samples per class. Then, these paired codes are input into a relation network [192]. The outputs are grouped by class, and parameterized by a probability distribution over latent codes $\mathbf{z}_n$ for class n in a low dimensional space. The decoder further generates a task-specifc model weights $\boldsymbol{\theta}_n$ for class n. The loss from the generated weights is propagated backward to update the model weights. In practice, generating a such high-dimensional set of parameters from a low-dimensional space

Algorithm 4 Bayesian MAML

```
Initialize Θ_0
for t = 1,... until convergence do
    Sample a batch of tasks B from p(T).
    for task T_j ∈ B do
        Calculate Θ^n_{T_j}(Θ_0) = SVGD_n(Θ_0; D^S_{T_j}, α).
        Calculate Θ^{n+s}_{T_j}(Θ_0) = SVGD_s(Θ^{n+s}_{T_j}(Θ_0); D^S_{T_j} ∪ D^Q_{T_j}, α).
    end for
    Θ_0 = Θ_0 − β∇_{Θ_0} Σ_{T_j∈B} d(Θ^n_{T_j}(Θ_0), Θ^{n+s}_{T_j}(Θ_0)).
end for
```

Algorithm 5 LLAML

```
Randomly initialize Θ.
while not converge do
    Sample a batch of tasks B from p(T).
    Estimate E_{(x_i,y_i)∼p_{T_j}}[−log P(y_i|x_i, θ)] using ML-LAPLACE in Algorithm 6.
    θ = θ − β∇_θ Σ_j E_{(x_i,y_i)∼p_{T_j}}[−log P(y_i|x_i, θ)].
end while
```

Algorithm 6 ML-LAPLACE

1: $\boldsymbol{\theta}_j^{'} = \boldsymbol{\theta}$.
2: **for** k = 1,...,K **do**
3: $\quad \boldsymbol{\theta}_j^{'} = \boldsymbol{\theta}_j^{'} + \alpha \nabla_{\boldsymbol{\theta}_j^{'}} \log P(y_i \in \mathcal{D}_{\mathcal{T}_j}^S | \boldsymbol{\theta}_j^{'}, \mathbf{x}_i \in \mathcal{D}_{\mathcal{T}_j}^S)$.
4: **end for**
5: According to [219], calculate curvature matrix $\hat{\mathbf{H}} = \nabla_{\boldsymbol{\theta}_j^{'}}^2 [-\log P(y_i \in \mathcal{D}_{\mathcal{T}_j}^Q | \boldsymbol{\theta}_j^{'}, \mathbf{x}_i \in \mathcal{D}_{\mathcal{T}_j}^Q)] + \nabla_{\boldsymbol{\theta}_j^{'}}^2 [-\log P(\boldsymbol{\theta}_j^{'} | \boldsymbol{\theta})]$.
6: **return** $-\log P(y_i \in \mathcal{D}_{\mathcal{T}_j}^Q | \boldsymbol{\theta}_j^{'}, \mathbf{x}_i \in \mathcal{D}_{\mathcal{T}_j}^Q) + \eta \log(\det(\hat{\mathbf{H}}))$.

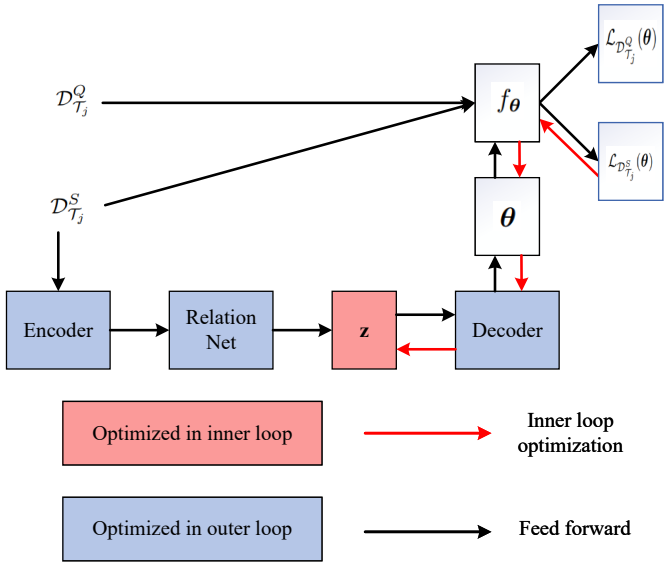


Fig. 18. Diagram of LEO.

is quite problematic. Thus, LEO uses pre-trained models, and only generates weights for the final layer, which limits the dimension of the model.

The key advantage of LEO is that it optimizes model weights in a lower dimensional latent embedding space, which improves generalization performance. However, it is more complex than that of MAML, because it needs to encode and decode the data samples.

7) iMAML: Because of the higher-order derivatives, MAML with Implicit Gradients (iMAML) was considered in [221] to deal with the issue of long optimization path in MAML caused by gradient degradation problems, such as vanishing and exploding gradients [222], [223]. Authors in [221] integrated regularization into the objective of MAML to guarantee appropriate learning while avoiding over-fitting. The objective of iMAML is formulated as

$$\min_{\boldsymbol{\theta}_i^{'}} \mathcal{L}(\boldsymbol{\theta}_i^{'}, \mathcal{D}_k) + \frac{\lambda}{2} \|\boldsymbol{\theta}_i^{'} - \boldsymbol{\theta}\|^2, \tag{118}$$

where λ is a scalar hyperparameter that controls the regularization strength. In (118), the regularization term $\frac{\lambda}{2}\|\boldsymbol{\theta}_i^{'} - \boldsymbol{\theta}\|^2$ encourages $\boldsymbol{\theta}_i^{'}$ to remain close to $\boldsymbol{\theta}$. The regularization strength λ plays an important role similar to the learning rate α in MAML, controlling the strength of prior model weights $\boldsymbol{\theta}$ relative to the dataset $\mathcal{D}_{\mathcal{T}}$. Ideally, the objective in (118) is solved by iteratively performing gradient descent to obtain the optimal $\boldsymbol{\theta}_i^{'}$. However, authors in [221] considered an implicit Jacobian to obtain $\boldsymbol{\theta}_i^{'}$ as

$$\frac{\partial \boldsymbol{\theta}_i^{'}}{\partial \boldsymbol{\theta}} = \left(\mathbf{I} + \frac{1}{\lambda} \nabla_{\boldsymbol{\theta}}^2 \mathcal{L}_i(\boldsymbol{\theta}_i) \right)^{-1}. \tag{119}$$

According to [224] and [225], Jacobian only depends on the final result of the algorithm, and not the path taken by the algorithm, thus, it effectively decouples the meta-gradient computation from the choice of inner loop optimizer.

iMAML significantly decreases memory costs because it does not need to store Hessian matrices like MAML, allowing for a higher flexibility in the selection of the inner loop optimizer. However, the computational costs are the same as that of MAML.

8) Online MAML: MAML assumes that a large set of tasks are available for meta training. However, in a practical system, tasks are likely available sequentially, which means that tasks may reveal one after the other. To deal with this issue, online meta learning was proposed in [226]. The objective of online meta learning is to minimize the regret, where the regret is defined as the difference between the loss of the meta-learner and the best performance achievable from online learning with non-convex loss functions [227]. This objective is captured by the regret over the entire sequence, and is denoted as

$$\text{Regret}_T = \sum_{t=1}^{T} \mathcal{L}_{\mathcal{T}_t}(\boldsymbol{\theta}_t^{'}) - \min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \mathcal{L}_{\mathcal{T}_t}(\boldsymbol{\theta}_t), \tag{120}$$

where $\sum_{t=1}^{T} \mathcal{L}_{\mathcal{T}_t}(\boldsymbol{\theta}_t^{'})$ reflects the accumulative loss calculated by the updated weights, and $\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \mathcal{L}_{\mathcal{T}_t}(\boldsymbol{\theta}_t)$ presents the minimum obtainable loss from a fixed set of initial model weights. The goal for the meta learner in (120) is to sequentially obtain model weights $\boldsymbol{\theta}_t^{'}$ that perform well on the loss sequence. To update $\boldsymbol{\theta}_{t+1}^{'}$, one of the simplest algorithms is following the leader (FTL) [228], [229], which updates the weights using

$$\boldsymbol{\theta}_{t+1}^{'} = \arg\min_{\boldsymbol{\theta}} \sum_{k=1}^{t} \mathcal{L}_{\mathcal{T}_k}(\boldsymbol{\theta}_k). \tag{121}$$

The gradient descent to perform meta update is given by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \beta \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}_k \sim p_t(\mathcal{T})} \mathcal{L}_{\mathcal{T}_k}(\boldsymbol{\theta}_k), \tag{122}$$

where $p_t(\mathcal{T})$ is a uniform distribution over tasks in the tth time slot, and β is the meta learning rate. In online meta learning, memory usage keeps increasing over time. This is because in each time slot, the incoming tasks and their corresponding datasets are stored in memory, which is used to obtain the model weights. The summary of gradient-based meta learning is summarized in Table VI.

D. Summary and Lessons Learned

In this section, we have reviewed three types of meta learning methodologies. We summarize the approaches along with references. From this review, we gather the following lessons learned:

TABLE VI
SUMMARY OF GRADIENT-BASED META LEARNING ALGORITHMS

Gradient-based Meta Learning	Advantage	Disadvantage	Condition
MAML [203]	Fine-tuned quickly Require fewer training samples	High computation complexity High computation latency High memory costs	Classification, regression and reinforcement learning
Meta-SGD [213]	Trainable learning rate Higher accuracy	Large number of model weights	Supervised learning Reinforcement learning
Reptile [214]	First-order simplification Low time and memory costs	Low convergence rate Low learning accuracy	Supervised learning Reinforcement learning
BMAML [215]	Learn multiple initializations	High memory costs	Supervised learning Reinforcement learning
LLAMA [218]	Bayesian interpretation Multiple solutions for a single task	High computaion costs	Supervised learning
LEO [220]	Optimize in lower dimensional space	High computation complexity Limited applicability to few-shot learning	Supervised learning Reinforcement learning
iMAML [221]	Low memory costs	High computation costs	Supervised learning Reinforcement learning
Online MAML [226]	Adapt to online learning	High computation complexity High computation latency High memory costs	Classification, regression and reinforcement learning

- Metric-based meta learning learn a feature space that is deployed to classification tasks based on input similarity scores. The advantages of metric-based meta learning are that (1) the approach of similarity-based classification is simple and (2) the computation latency at the testing phase is low when tasks are small, it is because the learning model does not need to make task-specific adjustments. However, when the tasks at the testing phase are distant from the tasks used in the training phase, the learning accuracy decreases. In addition, metric-based meta learning is usually used in supervised learning scenarios.
- In model-based meta learning, tasks are processed and represented in the state of the model-based system, which is then used to make classifications. Advantages of model-based meta learning are the flexibility of the internal dynamics of systems, and their broader applicability compared to metric-based meta learning. Unfortunately, the learning performance of model-based meta learning is worse than that of metric-based meta learning. Also, model-based meta learning is usually used in supervised learning scenarios.
- Gradient-based meta learning aims to learn new tasks quickly. The key advantage of gradient-based meta learning is that it can achieve much better learning performance on more task distributions than metric-based and model-based meta learning. However, gradient-based meta learning optimizes a meta-learner for each task which results in high computation and memory costs. In addition, gradient-based meta learning is usually used in supervised learning and reinforcement learning scenarios.
- Except for the theoretical convergence analysis of MAML, the convergence theory of other metric-based, model-based, and gradient-based meta learning algorithms is not derived. Therefore, detailed convergence analysis of meta learning is still need to be investigated.

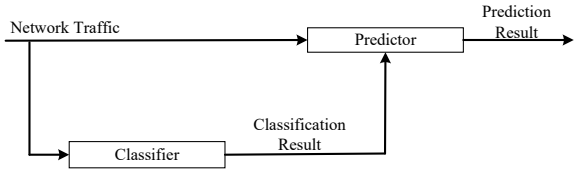


Fig. 19. Feed forward structure for traffic prediction.

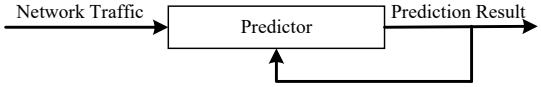


Fig. 20. Feed back structure for traffic prediction.

VI. META LEARNING IN WIRELESS COMMUNICATIONS

Due to the advantages of gradient-based meta learning, such as good generalization performance on new tasks and the model being easy to fine-tune, it has been widely used for optimization problems in wireless networks, such as traffic prediction [230], transmission rate maximization [231], [232], and multiple-input and multiple-output (MIMO) detectors [233].

A. Traffic Prediction

One of the traditional methods for network traffic prediction is a feed-forward predictor, which consists of a traffic classifier trained to recognize specific types of traffic, such as videos, web traffic, file downloading, and a predictor that takes the network traffic and classification results as inputs, as shown in Fig. 19. However, it requires a large amount of labeled datasets to train each traffic classifier, which leads to high computation complexity. To address this issue, authors in [230] proposed a feedback traffic prediction architecture based on a meta-learning scheme. The feed-back architecture is presented in Fig. 20, where the predictor is selected based on observed prediction accuracy by DRL, rather than the traffic class. The reason for using meta learning is that, it is recently employed

in robust adversarial learning, which can exploit models by taking advantage of obtainable model information and using it to create malicious attacks [234]–[236]. According to Fig. 21, the meta-learning scheme used in the predictor consists of a master policy and a set of sub-policies. The master policy is responsible for selecting which sub-policy is used for prediction during the next prediction interval. The meta-learning scheme in [230] allows for the updating of sub-predictors in real-time, so that the sub-predictors have the ability to adapt to variations in traffic patterns over time. The prediction accuracy of meta learning scheme significantly outperforms that of any single predictor. However, when a new sub-predictor is added to sub-predictors, the master policy needs to be retrained, which leads to high computational costs.

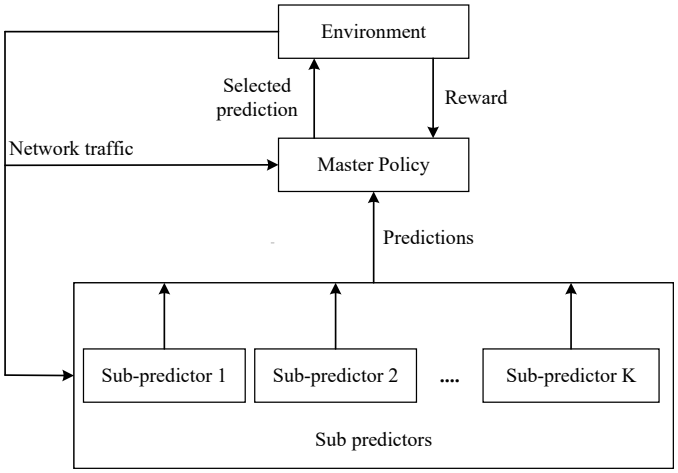


Fig. 21. Meta learning and DRL for traffic prediction.

B. Transmission Rate Maximization

The design of beamforming vectors that maximize the weighted sum rate (WSR) is an NP-hard problem and the iterative weighted minimum mean square error (WMMSE) is the most widely used technique to achieve optimal beamforming vectors. Although authors in [237]–[240] considered deep learning or graph neural networks (GNNs) to estimate the beamforming vectors, the sum rate performance of these methods was not higher than that of the WMMSE algorithm. An alternative method is to deploy meta-learning algorithms to solve the problem. Authors in [231] proposed a meta-learning-aided beamformer (MLBM) algorithm to solve the WSR maximization problem. The objective of MLBM is to minimize the global loss function $F(\mathbf{u}, \boldsymbol{w}, \mathbf{V})$, where $\mathbf{u}$, $\boldsymbol{w}$, and $\mathbf{V}$ are receiver gain vector, positive user weight vector, and transmit beamforming vector of all devices, respectively. Particularly, $F(\mathbf{u}, \boldsymbol{w}, \mathbf{V})$ is divided into three sub-problems $f(\mathbf{u})$, $f(\boldsymbol{w})$, and $f(\mathbf{v})$, and authors refer to the minimization of $F(\mathbf{u}, \boldsymbol{w}, \mathbf{V})$ as minimization of these three sub-problems $f(\mathbf{u})$, $f(\boldsymbol{w})$, and $f(\mathbf{v})$. A meta-learner neural network is deployed to treat these three sub-problems as three tasks and sequentially update $\mathbf{u}$, $\boldsymbol{w}$, and $\mathbf{V}$ of these three sub-problems, which is similar to the inner loop of task-specific adaptation of gradient-based meta-learning. Then, the updated $\mathbf{u}$, $\boldsymbol{w}$, and $\mathbf{V}$ are used to calculate global loss function $F(\mathbf{u}, \boldsymbol{w}, \mathbf{V})$, and update the learning weights of the meta learner, which is similar to the outer loop of meta initialization training of gradient-based meta-learning. Through iteratively updating $\mathbf{u}$, $\boldsymbol{w}$, $\mathbf{V}$, and meta-learner weights, MLBM can achieve a higher transmission rate than that of the WMMSE algorithm particularly in the high SNR regime, and achieves a similar performance when SNR is small. In addition, authors in [232] compared meta learning and transfer learning in beamforming design, and verified that meta learning was able to provide a higher transmission rate compared to that of transfer learning.

C. MIMO Detectors

Deep neural networks (DNNs) have the potential for efficiently balancing the bit-error rate minimization and computation complexity of MIMO detectors [241]–[244]. Unfortunately, the existing DNN-based MIMO detectors are difficult to be deployed in practical systems due to their slow convergence speed and low robustness in new environments. To deal with this issue, authors in [233] proposed meta-learning-based MIMO detectors, which could be used in channel-sensitive environments. In particular, an expectation propagation (EP) for signal detection is unfolded as EPNet, and damping factors are set as trainable parameters to adapt to new channels [245]–[248]. Damping factors are relevant to channel statistics. To train the damping factors, a large amount of labeled channel state information (CSI) is required. However, in a dynamic real-time wireless system, it is impractical to obtain enough CSI to train the damping factors. Thus, meta learning is deployed to update the damping factors efficiently by using a small training set so that they can quickly adapt to new environments.

D. Summary and Lessons Learned

In this section, we have reviewed three research directions of meta learning over wireless networks. We summarize the approaches along with references. From this review, we gather the following lessons learned:

- Meta learning is effective for wireless networks with multiple tasks or sub-problems, and achieves better learning and network transmission performance, such as higher learning accuracy and transmission rate, compared with conventional optimization methods.
- For the research works we have discussed in this section, the proposed scheme in each work can quickly adapt to new environments with the help of meta learning. However, the computation and memory costs of meta learning are much higher than that of traditional optimization approaches.
- Although meta learning has been widely applied in the design of signal processing and network management [230]–[233], [249]–[251], its applications in wireless networks still face several challenges. First, multiple meta learning models can be generated at the base stations. Sophisticated model selection schemes are still unknown to adapt to different learning tasks. Second, the meta learning models are transmitted to devices by using extra

radio resources, which has a heavy burden on wireless networks [252]. Therefore, it is a dilemma to balance the learning performance and the communication costs of meta learning.

- To the best of the authors' knowledge, no research works focus on how wireless factors affect meta learning. Therefore, we directly introduce how meta learning is used to solve the proposed optimization problems in wireless networks. Also, the convergence analysis of meta learning over wireless networks is not investigated.

VII. Federated Meta Learning

Gradient-based meta learning algorithms, such as MAML, are well known for their rapid adaptation and good generalization to multiple learning tasks, which makes them particularly suitable for federated settings where the decentralized training data is non-IID and highly personalized [208], [253]–[255]. In this section, federated meta learning methodologies and their applications over wireless networks are discussed.

A. FedMeta Methodologies

In this subsection, we introduce three main research directions of FedMeta methodologies, including MAML/Meta-SGD-based FedMeta, Collaborative FedMeta, and ADMM-FedMeta. The introduced FedMeta methodologies are fundamental FedMeta algorithms from the CS community and the transmission between servers and devices is error-free without considering any wireless factors.

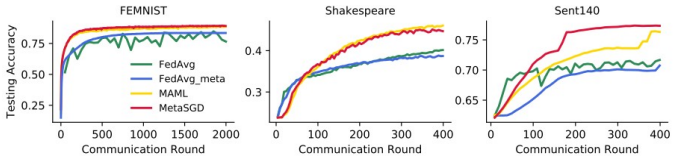


Fig. 22. Comparison of the convergence speed of FedMeta and FedAvg from [256].

1) MAML/Meta-SGD-based FedMeta: The federated meta learning framework was first proposed in [256], where authors integrated MAML and Meta-SGD [213] into FL. The objective of the algorithm is to collaboratively meta-train an algorithm using datasets from distributed devices. For model aggregation in the server, as shown in **Algorithm 7**, the server maintains the initialization parameters $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$, updates them through the testing loss from the selected devices, and transmits them to the selected devices. For the local model training and testing in devices, as shown in **Algorithm** 8, first, the kth device trains the learning weights $\boldsymbol{\theta}$ obtained from the server using its support dataset D_S^k. Second, the kth device tests the trained learning model, calculates the testing loss $\mathcal{L}_{D_Q^u}(\boldsymbol{\theta})$ based on its query set D_Q^u, and transmits $\mathcal{L}_{D_Q^u}(\boldsymbol{\theta})$ to the server. For Meta-SGD, the vector $\boldsymbol{\alpha}$ is also delivered to the server as part of the algorithm parameters which are used for parameter updating. The detailed FedMeta with MAML and Meta-SGD at the server, and model training of MAML or Meta-SGD at the device are introduced in **Algorithm 7** and **Algorithm 8**, respectively. According to [256], the comparison of the convergence speed of FedMeta and FedAvg is shown in Fig. 22. It is observed that FedMeta provides a faster convergence speed and higher learning accuracy.

Algorithm 7 FedMeta with MAML and Meta-SGD at the server

1: Initialize step size hyperparameters α and β, randomly initialize learning weights $\boldsymbol{\theta}$.
2: **for** each time slot t=1,2,... **do**
3: Sample K devices, and distribute $\boldsymbol{\theta}$ for MAML or $(\boldsymbol{\theta}, \boldsymbol{\alpha})$ for Meta-SGD to these K devices.
4: **for** the kth device in K devices **do**
5: Obtain testing loss $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{D_Q^k}(\boldsymbol{\theta}_k)$ from the model training of the MAML.
6: Obtain testing loss $\nabla_{(\boldsymbol{\theta},\boldsymbol{\alpha})} \mathcal{L}_{D_Q^k}(\boldsymbol{\theta}_k)$ from the model training of the Meta-SGD.
7: **end for**
8: Update $\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\beta}{K} \sum_{k=1}^{K} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{D_Q^k}(\boldsymbol{\theta}_k)$ for the MAML.
9: Update $(\boldsymbol{\theta}, \boldsymbol{\alpha}) = (\boldsymbol{\theta}, \boldsymbol{\alpha}) - \frac{\beta}{K} \sum_{k=1}^{K} \nabla_{(\boldsymbol{\theta},\boldsymbol{\alpha})} \mathcal{L}_{D_Q^k}(\boldsymbol{\theta}_k)$ for the MetaSGD.
10: **end for**

Algorithm 8 Model Training for MAML and MetaSGD

1: Sample support set D_S^k and query set D_Q^k of the kth device.
2: $\mathcal{L}_{D_S^k}(\boldsymbol{\theta}) = \frac{1}{|D_S^k|} \sum_{(x,y)\in D_S^k} l(f_{\boldsymbol{\theta}}(x), y)$.
3: $\boldsymbol{\theta}_k = \boldsymbol{\theta} - \alpha \nabla \mathcal{L}_{D_S^k}(\boldsymbol{\theta})$ for the MAML and $\boldsymbol{\theta}_k = \boldsymbol{\theta} - \boldsymbol{\alpha} \circ \nabla \mathcal{L}_{D_S^k}(\boldsymbol{\theta})$ for the MetaSGD.
4: $\mathcal{L}_{D_Q^k}(\boldsymbol{\theta}_k) = \frac{1}{|D_Q^k|} \sum_{(x',y')\in D_Q^k} l(f_{\boldsymbol{\theta}_k}(x'), y')$.
5: Transmit $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{D_Q^k}(\boldsymbol{\theta}_k)$ and $\nabla_{(\boldsymbol{\theta},\boldsymbol{\alpha})} \mathcal{L}_{D_Q^k}(\boldsymbol{\theta}_k)$ to the server.

2) Collaborative FedMeta: Authors in [257] proposed a platform-based collaborative learning framework, where a model was first trained in a set of edge nodes by FedMeta, and then it was rapidly adapted to learn a new task at the target edge node with a few data samples. This can deal with the constrained computing resources and limited local data issues of each edge node. The FedMeta algorithm used in [257] is the same as that used in [256]. However, according to Fig. 23, the main differences are that: 1) Authors in [257] deployed a set of source edge nodes only with a support dataset to train the local models, rather than authors in [256] assuming that each device had both support and query datasets to train and test the local model; 2) Authors in [257] assumed that each source edge node focused on only one task to train the local model, rather than authors in [256] assuming that each device had multiple tasks; 3) Authors in [257] used a platform to aggregate local models from all source edge nodes and transmitted the aggregated model to the target edge node for new task adaptation, rather than authors in [256] assuming that each device was able to adapt to new tasks. With model training at multiple edge nodes, FedMeta can adapt to multiple tasks in parallel. However, for cases with a large number of tasks, delivering multiple learning models simultaneously can result in high transmission latency.

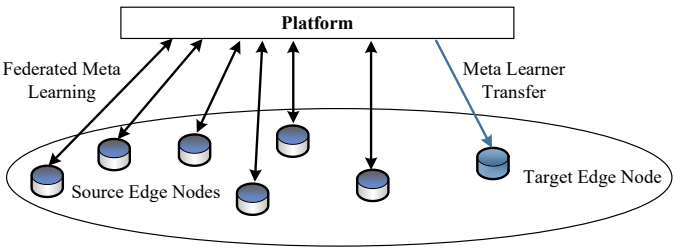


Fig. 23. Platform-based collaborative federated meta learning framework.

3) ADMM-FedMeta: Based on [257], authors in [258] proposed an ADMM-based algorithm, called ADMM-FedMeta, to decompose the initial optimization problem into several subproblems which can be solved in parallel across edge nodes and the platform. First, authors still applied the platform-based FedMeta architecture in [257] to enable edge nodes to collaboratively learn a meta-model with the knowledge transfer of previous tasks. Then, the FedMeta problem is defined as a regularized optimization problem, where the previous knowledge is extracted as regularization, and the optimization problem is denoted as

$$\min_{\boldsymbol{\theta}_i,\boldsymbol{\theta}} \sum_{k\in\mathcal{I}} \frac{D_k}{\sum_{k\in\mathcal{I}} D_k} \mathcal{L}_k(\phi_k(\boldsymbol{\theta}_k), \mathcal{D}_k^q) + \lambda D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_p), \quad (123)$$
$$\text{s.t.} \quad \boldsymbol{\theta}_k - \boldsymbol{\theta} = \mathbf{0}, k \in \mathcal{I},$$

where $\phi_k(\boldsymbol{\theta}_k)$ is written as (104), $D_h(\boldsymbol{\theta}, \boldsymbol{\theta}_p)$ is a regularization parameter that can extract the valuable knowledge from the prior model to facilitate fast edge training and alleviate catastrophic forgetting [259]. In (123), $\boldsymbol{\theta}_p$ is the prior model weights, λ is a penalty parameter that is used to balance the trade-off between the loss and regularization, $\mathcal{I}$ is the set of edge nodes, $\mathcal{D}_k$ is the dataset of the kth ($k \in \mathcal{I}$) edge node, D_k is the number of data samples in $\mathcal{D}_k$, and $\mathcal{D}_k$ is divided into two disjoint datasets, i.e., the support set $\mathcal{D}_k^s$ and query set $\mathcal{D}_k^q$. Through penalizing variations in the model via regularization, the learned model from (123) is close to the prior model for enabling collaborative edge learning without forgetting prior knowledge, so that the learned meta model can adapt to different tasks. To solve the optimization problem in (123), the augmented Lagrangian function is deployed, which is written as

$$\mathcal{L}(\{\boldsymbol{\theta}_k, \boldsymbol{w}_k\}, \boldsymbol{\theta}) = \sum_{k\in\mathcal{I}} \Big(\frac{D_k}{\sum_{k\in\mathcal{I}} D_k} \mathcal{L}_k(\phi_k(\boldsymbol{\theta}_k), \mathcal{D}_k^q)$$
$$+ < \boldsymbol{w}_k, \boldsymbol{\theta}_k - \boldsymbol{\theta} > + \frac{\rho_k}{2} \|\boldsymbol{\theta}_k - \boldsymbol{\theta}\|^2 \Big) + \lambda D_h(\boldsymbol{\theta}_k, \boldsymbol{\theta}), \quad (124)$$

where $\boldsymbol{w}_k$ is a dual variable and $\rho_k > 0$ is a penalty parameter. To optimize $\boldsymbol{\theta}_k$ $\boldsymbol{\theta}$, and $\boldsymbol{w}_k$, ADMM method is applied [260]–[269]. The traditional ADMM decomposes the optimization in (110) into a set of sub-problems that can be solved in parallel, which means that calculating $D_h(\boldsymbol{\theta}_k, \boldsymbol{\theta})$ and $\mathcal{L}_k(\phi_k(\boldsymbol{\theta}_k), \mathcal{D}_k^q)$ separately. Thus, the vector $\boldsymbol{\theta}_k$, $\boldsymbol{\theta}$, and $\boldsymbol{w}_k$ are updated alternatively as follows

$$\boldsymbol{\theta}^{t+1} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\{\boldsymbol{\theta}_k^t, \boldsymbol{w}_k^t\}, \boldsymbol{\theta}), \quad (125)$$

$$\boldsymbol{\theta}_k^{t+1} = \arg\min_{\boldsymbol{\theta}_k} \mathcal{L}_k(\boldsymbol{\theta}_k, \boldsymbol{w}_k^t, \boldsymbol{\theta}^{t+1}), \quad (126)$$

and

$$\boldsymbol{w}_k^{t+1} = \boldsymbol{w}_k^t + \rho_k(\boldsymbol{\theta}_k^{t+1} - \boldsymbol{\theta}^{t+1}). \quad (127)$$

In (125), (126), and (127), the platform and each device select the weights that can achieve the minimum loss, and update the dual variable by the difference of local and global weights. Based on (125), (126), and (127), the updating strategy is 1) updating $\boldsymbol{\theta}$ at the platform and 2) updating $\{\boldsymbol{\theta}_k, \boldsymbol{w}_k\}$. The advantage of using ADMM-FedMeta is that the decoupled subproblems can be allocated to multiple edge nodes to solve simultaneously, which helps to alleviate the local computational costs and improve the computation efficiency. According to [258], the comparison of the convergence speed of ADMM-FedMeta and FedAvg is shown in Fig. 24. It is observed that ADMM-FedMeta achieves a much higher convergence speed than that of FedAvg.

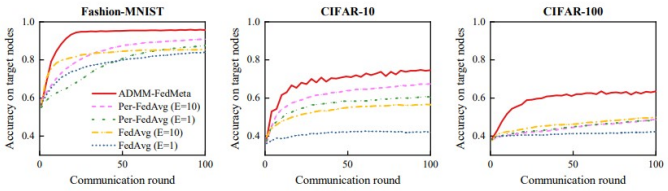


Fig. 24. Comparison of the convergence speed of ADMM-FedMeta and FedAvg from [258].

B. FedMeta in Wireless Networks

FedMeta integrates the advantages of FL and meta learning, which enables local model sharing without privacy issues and fast adaptation to new tasks. Through learning an initial shared model, devices can quickly adapt the learned model to their local datasets via one or a few gradient descent steps. Despite its advantages, FedMeta still has several challenges: First, the number of participating devices can be enormous. When devices are randomly selected, it can lead to a low convergence speed [258]. Second, the convergence performance of FedMeta in a wireless network is highly related to its latency, which includes computation latency, determined by the size of local datasets and CPU types of devices, and transmission latency, determined by channel gains, interference, and transmission power [270]. If these factors are not optimized, high latency can result in unexpected training delay and communication inefficiency [271]. In this subsection, we introduce device selection and energy efficiency of FedMeta over wireless networks, based on the fundamental FedMeta methodologies.

1) Device Selection: To deal with high training delay and communication inefficiency, authors in [272] developed a FedMeta with a non-uniform device selection scheme to accelerate the convergence, and rigorously analyzed the contribution of each device to the global loss reduction in each time slot. Then, a resource allocation problem integrating FedMeta into multi-access wireless systems is proposed to jointly improve the convergence rate and minimize the latency along with energy cost. The learning structure of FedMeta proposed in [272] is

the same as that in [256]. The device selection problem in the tth time slot is formulated as

$$\begin{aligned} \max_{z_k} \ & \sum_{k\in\mathcal{N}} z_k u_k^t \\ \text{s.t.} \ & \sum_{k\in\mathcal{N}} z_k = n_k \\ & z_k \in \{0,1\}. \end{aligned} \tag{128}$$

In (128), z_k is a binary variable, if $z_k = 1$, the kth device is selected, otherwise, not, and $\mathcal{N}$ is the set of devices. u_k^t is the contribution of the kth device to the convergence in the tth time slot, defined as

$$u_k^t = \sum_{i=0}^{\tau-1} \|\nabla F_k(\boldsymbol{\theta}_k^{i,t})\|^2 - 2(\lambda_1 + \frac{\lambda_2}{\sqrt{D_k}})\|\nabla F_i(\boldsymbol{\theta}_k^{i,t})\|, \tag{129}$$

where $F_k(\boldsymbol{\theta}_k^{i,t})$ is a meta function of the kth device, τ is the number of iterations of gradient descent, D_k is the number of data samples in the kth device, and λ_1 and λ_2 are position constants. The detailed proof of u_k^t is presented in Appendix J of the technical report [273]. To apply FedMeta over wireless networks, the authors propose a resource allocation problem, capturing the trade-offs among the convergence, computation and communication latency, and energy consumption. Then, the optimization problem is decomposed into two sub-problems. The first sub-problem aims at controlling the CPU-cycle frequencies for devices to minimize energy consumption and computation latency. The second sub-problem controls transmission power and resource block allocation to maximize the convergence speed while minimizing the transmission cost and latency. Both of the sub-problems are solved by KKT conditions.

2) Energy Efficiency: In FedMeta, each task is owned by a device, and each updating iteration needs to communicate with the server. In each iteration, each device updates its local model and transmits it to the server, where the meta model is updated in a global step and then the updated meta model is feedback to all devices. Since these procedures involve local computation and communication energy consumption, it is important to minimize the computation costs and save energy for communication, especially for a device with limited computation capability and energy. To minimize energy and computation costs when performing FedMeta, authors in [274] considered an energy-efficient FedMeta framework, where a meta-backward algorithm was proposed, to learn a meta model with low computation and communication energy consumption. In the backward manner, in the kth step, $\boldsymbol{\theta}_i^k$ is computed as

$$\boldsymbol{\theta}_i^k = \boldsymbol{\theta}_i^{k+1} + \alpha \nabla_{\boldsymbol{\theta}_i^k} \mathcal{L}(\boldsymbol{\theta}_i, \mathcal{D}_i). \tag{130}$$

However, when computing in a backward manner, the term $\nabla_{\boldsymbol{\theta}_i^k} \mathcal{L}(\boldsymbol{\theta}_i, \mathcal{D}_i)$ cannot be calculated. To address this issue, $\nabla_{\boldsymbol{\theta}_i^k} \mathcal{L}(\boldsymbol{\theta}_i, \mathcal{D}_i)$ can be replaced by $\nabla_{\boldsymbol{\theta}_i^{k+1}} \mathcal{L}(\boldsymbol{\theta}_i, \mathcal{D}_i)$, because they are close to each other under smoothness assumption. In the kth backward step, to find the optimal $\boldsymbol{\theta}_i^k$, the optimization problem is fomulated as

$$\begin{aligned} \min_{\{\boldsymbol{\theta}_i\}_{i=1}^N} \ & \sum_{i=1}^{N} \Big(\mathcal{L}(\boldsymbol{\theta}_i) - \mathcal{L}(\boldsymbol{\theta}_i^{k,0})\Big)^2, \\ \text{s.t.} \ & \|\boldsymbol{\theta}_i - \boldsymbol{\psi}^{k+1}\|^2 \le \delta_k, \end{aligned} \tag{131}$$

where $\boldsymbol{\psi}^{k+1} = \frac{1}{N}\sum_{i=1}^N \boldsymbol{\theta}_i^{k+1}$, N is the total number of tasks, $\delta_k \to 0$, and

$$\boldsymbol{\theta}_i^{k,0} = \boldsymbol{\theta}_i^{k+1} + \alpha \nabla_{\boldsymbol{\theta}_i^{k+1}} \mathcal{L}(\boldsymbol{\theta}_i^{k+1}, \mathcal{D}_i). \tag{132}$$

Obviously, based on the constraint in (131), we need to find the optimal $\boldsymbol{\theta}_i^k$ which is close to the average weight $\boldsymbol{\psi}$ among all tasks. To do so, a projection gradient descent (PGD) is introduced to solve the problem in (131). Note that PGD is a standard way to solve constrained optimization problems [275], [276]. The projection problem for the ith task is written as

$$\begin{aligned} \min_{\boldsymbol{\theta}_i^k} \ & \frac{1}{2}\|\boldsymbol{\theta}_i^k - \boldsymbol{\theta}_i^{k,0}\|^2, \\ \text{s.t.} \ & \|\boldsymbol{\theta}_i^k - \boldsymbol{\psi}^{k+1}\|^2 \le \delta_k. \end{aligned} \tag{133}$$

The Lagrangian function of (133) is denoted as

$$L(\boldsymbol{\theta}_i, \boldsymbol{\psi}, \mu) = \frac{1}{2}\|\boldsymbol{\theta}_i^k - \boldsymbol{\theta}_i^{k,0}\|^2 + \mu(\|\boldsymbol{\theta}_i^k - \boldsymbol{\psi}^{k+1}\|^2 - \delta_k). \tag{134}$$

Using KKT conditions, we can solve the problem in (133). The proposed meta-backward algorithm is computationally efficient, as it has a closed-form solution calculated by KKT conditions in each iteration.

C. Convergence Analysis of FedMeta

For FedMeta methodologies, authors usually derive the upper bound of $\{\mathcal{L}(\boldsymbol{\theta}^T) - \mathcal{L}(\boldsymbol{\theta}^*)\}$ and $\|\nabla f(\boldsymbol{\theta}^*)\|$ of FedMeta and ADMM-FedMeta for convergence analysis, respectively. Except from the basic assumptions 1 - 4 introduced in Section III, another assumption is introduced in the following.

Assumption 6. Each loss functions $\mathcal{L}_n(\boldsymbol{\theta})$ is μ-strongly convex for any $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^{'}$:

$$\langle \nabla \mathcal{L}_n(\boldsymbol{\theta}) - \nabla \mathcal{L}_n(\boldsymbol{\theta}^{'}), \boldsymbol{\theta} - \boldsymbol{\theta}^{'} \rangle \ge \mu \|\boldsymbol{\theta} - \boldsymbol{\theta}^{'}\|^2, \tag{135}$$

where μ is a positive constant. The upper bound of $\{\mathcal{L}(\boldsymbol{\theta}^T) - \mathcal{L}(\boldsymbol{\theta}^*)\}$ of FedMeta in [257] is derived as

$$\mathcal{L}(\boldsymbol{\theta}^T) - \mathcal{L}(\boldsymbol{\theta}^*) \le \xi^T [\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^*)] + \frac{G(1-\alpha\mu)}{1-\xi^{T_0}} h(T_0), \tag{136}$$

where α is learning rate, G is defined in assumption 3, ξ is a constant determined by learning rate, μ, and L defined in assumption 2, T_0 is the duration of one local update step, and $T = NT_0$ is a fixed duration given the number of local update steps N. The term $\frac{G(1-\alpha\mu)}{1-\xi^{T_0}} h(T_0)$ captures the error introduced by both task dissimilarity and multiple local updates through the function $h(T_0)$. In (136), $h(T_0)$ indicates how the task similarity and T_0 impact the convergence performance, namely, given a fixed duration T, the convergence error decreases with the task similarity while increasing with the number of local update steps when T_0 is large. Therefore,

the platform-based FedMeta is able to balance between the platform-edge communication cost and the local computation cost by controlling the number of local update steps per communication round, depending on the task similarity across the edge nodes. Furthermore, for the convergence analysis of ADMM-FedMeta in [258], it is proved that $\boldsymbol{\theta}^t$ has at least one limit point and each limit point $\boldsymbol{\theta}^*$ has a stationary solution, namely, $\|\nabla f(\boldsymbol{\theta}^*)\| = 0$.

Note that there are no research works obtained the theoretical convergence analysis of FedMeta with device selection, resource allocation, and energy consumption over wireless networks.

D. Summary and Lessons Learned

In this section, we have reviewed three research directions of FedMeta methodologies and two research directions of FedMeta over wireless networks. We summarize the approaches along with references. From this review, we gather the following lessons learned:

- FedMeta is the integration of FL with meta learning, which not only guarantees user privacy but also quickly adapts to multiple tasks. However, for scenarios with plenty of tasks, optimizing a meta-learner for each task and frequently delivering local and global models between servers and devices lead to high computation and communication latency.
- Given that FedMeta is a specialization of FL, FedMeta has all the challenges that exist in FL, plus additional challenges. The key challenge is how to effectively train models across multiple devices. Architecture challenge includes whether it is possible to use a peer-to-peer architecture. Another challenge would arise if the system should not only focus on the globally best algorithm for a task (an entire dataset) but if per-instance algorithm selection should be learned. This would make the whole system even more complex.
- For the research works we have discussed in this section, a large number of devices participating in FedMeta results in lower convergence rate, higher communication latency, higher energy costs, and system heterogeneity than that of FL and meta learning. Device selection and energy-efficient FedMeta cannot achieve significant effectiveness. Therefore, the trade-off between the local model updating and global model aggregation in FedMeta to minimize the convergence time and energy cost from a long-term perspective needs to be investigated. Also, how to characterize the convergence properties and communication complexity of FedMeta over wireless networks requires further research.

VIII. Implementation Platforms

In this section, implementation platforms of FL, meta learning, and FedMeta are introduced.

A. FL Platforms

Based on the introduced research works of FL methodologies and their applications over wireless networks, several platforms for FL are constructed, which are PySyft, TensorFlow Federated (TFF), Federated AI Technology Enabler (FATE), Tensor/IO, Functional FL in Erlang (FFL-ERL), CrypTen, and LEAF.

- **PySyft:** PySyft is an open-source multi-language library enabling secure and private machine learning by wrapping and extending popular deep learning frameworks such as PyTorch in a transparent, lightweight, and user-friendly manner. Its aim is to both help popularize privacy-preserving techniques in machine learning by making them as accessible as possible by Python bindings and common tools familiar to researchers and data scientists, as well as to be extensible such that new FL, Multi-Party Computation, or Differential Privacy methods can be flexibly and simply implemented and integrated [277]. PySyft decouples private data from model training, using FL within PyTorch.
- **TFF:** TensorFlow Federated (TFF) is an open-source framework for machine learning and other computations on decentralized data. TFF has been developed to facilitate open research and experimentation with FL. TFF enables developers to use the included federated learning algorithms with their models and data, as well as to experiment with novel algorithms. The building blocks provided by TFF can also be used to implement non-learning computations, such as aggregated analytics over decentralized data [278].
- **FATE:** FATE is an open-source project initiated by Webank's AI Department to provide a secure computing framework to support the federated AI ecosystem. It implements multiple secure computation protocols to enable big data collaboration with data protection regulation compliance [279]. Furthermore, FATE is able to support various FL architectures and ML algorithms.
- **Tensor/IO:** Tensor/IO is a platform that brings TFF to mobile devices such as iOS and Android [280]. Although this platform cannot implement any ML algorithms, the platform cooperates with TFF to ease the implementation and deployment of ML algorithms on mobile phones.
- **FEL-ERL:** The functional programming language Erlang is well-suited for concurrent and distributed applications, which is suitable for establishing real-time systems. FEL-ERL was proposed by Gregor Ulm, Emil Gustavsson, and Mats Jirstran in [281]. They evaluated FEL-ERL in two scenarios: one in which the entire system has been written in Erlang, and another in which Erlang is relegated to coordinating device processes that rely on performing numerical computations in the programming language C. The authors found that Erlang incurs a performance penalty, but for certain use cases this may not be detrimental, considering the trade-off between speed of development (Erlang) versus performance (C).
- **CrypTen:** CrypTen is a new framework built on PyTorch to facilitate research in FL [282]. There are a few benefits to using this platform. One benefit is that CrypTen enables machine learning researchers, who may not be cryptography experts, to easily experiment with FL. Another benefit is that the platform is made

with real-world challenges in mind. Therefore, it has the potential to be applicable to large number of real-world applications.

- **LEAF:** LEAF is a modular framework for FL, multi-task learning, meta learning, and FedMeta. LEAF was proposed in [283] which allowed researchers to reason about new proposed solutions under more realistic assumptions than previous benchmarks. LEAF is kept updating with new datasets, metrics, and open-source solutions to foster informed and grounded progress in ML field.

B. Meta Learning and FedMeta Platforms

Based on the introduced research works of meta learning/FedMeta methodologies and their applications over wireless networks, LEAF platform has been used for these two learning algorithms' implementation, which has already been introduced in the previous section. Recently, a novel meta learning platform called Awesome-META+ is created.

- **Awesome-META+:** Awesome-META+ provides a comprehensive and reliable meta learning framework code that can adapt to multiple domains and improve academic research efficiency. Furthermore, it provides a convenient and simple model deployment solution to lower the threshold and promote the development of meta-learning and its transfer fields. In addition, it provides a comprehensive and complete information summary and learning platform for the meta learning field to stimulate the vitality of the meta-learning community [284].

IX. Real-World Applications

In this section, real-world applications for FL, meta learning, and FedMeta in real scenarios are introduced.

A. FL

FL is used in Google keyboard, intelligent medical diagnosis system, and autonomous driving vehicles.

1) Google Keyboard: Google started a project in 2016 to implement FL among mobile devices to improve the learning accuracy of keyboard input prediction, while guaranteeing the privacy of users simultaneously [285]. Through developing language models, the recommendation system is also promoted. Also, it can be extended to other recommendation applications by integrating with FL. When mobile devices send requests, the corresponding suggestions are quickly provided by learning models.

2) Intelligent Medical Diagnosis System: Different hospitals own the electronic health records of different patient populations and these records are difficult to share across hospitals because of the protection of patient privacy. This creates a big barrier to developing effective analytical approaches that are generalizable. Fortunately, FL is able to use interconnected medical systems to transform medical data into diagnostic evidence to assist doctors in forward-looking diagnosis of patients [286].

3) Autonomous Driving Vehicles: The autonomous driving system is a complicated system with a large amount of data. With multiple task layers from sensing, object detection, and tracking, to external object movement intention estimation, driving decision, and actuation, designing an autonomous management system requires real-time dynamics capturing and a huge amount of data from devices other than itself. Traditional centralized ML in autonomous driving aggregates large-scale data from all devices to a central server, which results in high computation and communication latency. Fortunately, FL sufficiently utilizes the computing capabilities of multiple learning agents to improve learning efficiency while providing better privacy for vehicles. Also, FL empowers adaption to environment dynamics, providing feature learning in different geographical locations, weather conditions, and pedestrians behavior dynamics [287].

B. Meta Learning

Meta learning is used in highly automated AI, natural language processing, and robotics.

1) Automated AI: In recent years, with the increasing amount of data, data scientists cannot address all challenging tasks in ML due to a lack of expertise and experience in the respective domain. Fortunately, automated AI is a data mining-based formalism that aims to reduce human effort and speed up the development cycle through automation. Based on the context of automated AI, one main advantage of meta learning techniques is that they allow hand-engineered algorithms to be replaced with novel automated methods which are designed in a data-driven way [288].

2) Natural Language Processing: Deep learning is one of the mainstream techniques in the NLP area and creates significant performance in many NLP problems. However, deep learning models are data-hungry, which limits learning models' application to different NLP tasks because collecting in-genre data for model training is costly. To address this issue, meta learning is deployed to the NLP area to learn more general NLP models, including better parameter initialization, optimization strategy, network architecture, distance metrics, and beyond [289].

3) Robotics: In the real world, a robot may encounter any situation from motor failures to finding itself in a rocky terrain where the dynamics of the robot are significantly different from one to another. As a result, the ability to adapt rapidly to unforeseen situations is one of the main open challenges for robotics. To solve the problem, meta learning is considered, where the robot enable to adapt to the current situation with only a few gradient steps by using a single set of meta-trained parameters as initial parameters [290].

C. FedMeta

FedMeta integrates the advantages of FL and meta learning, and it can be used in autonomous driving vehicles, automated AI, NLP, and robotics, which have been introduced in previous sections.

X. Open Problems and Future Directions

Given the general research areas and challenges in integrating FL, meta learning, and FedMeta methodologies and their applications over wireless networks, we discuss the remaining research problems and future directions. The ultimate goal is to seamlessly integrate FL/meta-learning/FedMeta with wireless networks, resulting in a wide range of new designs ranging from techniques for computation offloading to network architectures.

A. Federated Learning

1) FL Methodologies:

- **Unreliable Model Upload:** In FL, devices may deliver low-quality local models trained by malicious data samples, which can adversely affect the learning accuracy. To address this issue, a reputation metric can be deployed to measure the reliability of each device, and select reliable devices for model aggregation.
- **Systematic and Model Heterogeneity:** A large number of devices and hardware specifications bring systematic heterogeneity to FL in practical systems. Also, FL has coupled with many different learning paradigms, which brings model heterogeneity. To deal with systematic heterogeneity, multi-center FL can be considered, where the devices with similar heterogeneity are clustered into a group for model aggregation. Meanwhile, to deal with model heterogeneity, one possible solution is to learn a personalized model for each device, so-called on-device personalization [291]. Its goal is to train a model for each device, based on the dataset of each device.
- **Imbalanced Data:** FL mainly focuses on IID and non-IID data. However, the data of real-world applications, such as computer vision and biomedicine, follows an imbalanced distribution [292], [293]. Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally [294]. To deal with this issue, a monitor scheme is integrated into FL that can infer the composition of training data in each FL round and detect the existence of possible global imbalance [295].
- **Unsupervised FL:** Supervised FL enables multiple devices to share the trained model without sharing their labeled data. However, in real-world applications, the observed data may be unlabeled, which could limit the applicability of FL. To address this issue, a federation of unsupervised learning (FedUL) was proposed in [95], where the unlabeled data were transformed into surrogate labeled data for each device, a modified model was trained by supervised FL, and the unsupervised FL learning model was recovered from the modified model. However, FedUL is not suitable for non-IID data. Thus, FedUL can be further improved by integrating with advanced FL aggregation or optimization schemes for non-IID settings [296].
- **Federated Reinforcement Learning:** In FRL, rather than just uploading and downloading models, the agents need to exchange intermediate results and observations between themselves or with a central server. However, because of limited communication resources, the overhead is high, especially with an increasing number of agents. Meanwhile, some deep reinforcement learning (DRL) algorithms, such as deep Q network (DQN) [297] and Deep Deterministic Policy Gradient (DDPG) [298], have multiple layers or networks, which contain millions of parameters, resulting in extremely high overhead. To solve these issues, several research directions should be considered. First, dynamic global model methods need to be designed to optimize the number of model exchanges. Second, devices or agents need to exchange the important parts of models or observations.
- **Robust to Poisoned Data:** FL models are usually trained by non-poisoned data. However, in practical scenarios, malicious servers or devices have negative effects on model training. Although FL by itself has a certain level of resilience against attacks, the frequent connections between servers and devices may spread the risk over networks and reduce the learning performance. Therefore, how to design an FL algorithm that is robust to poisoned data and design resilient networks for FL to avoid spreading attacks needs to be investigated.

2) FL over Wireless Networks:

- **Learning Convergence Analysis:** One of the most important considerations of FL is the convergence performance. Most existing research works in [148], [156], [164], [299]–[301] deployed traditional optimization methods to optimize wireless factors, such as transmission scheduling, transmission error, and energy to analyze the convergence of FL, and assumed that the optimization problem was convex. However, FL over practical wireless networks may not satisfy these conditions, especially when the optimization problems are non-convex. Also, the performance of convergence can be affected by dynamic wireless channels and device mobility. To address these issues, one possible solution is to deploy an FL algorithm that can handle heterogeneous device datasets, and capture the trade-off between convergence and energy consumption of devices with heterogeneous computing and power resources [270].
- **Device Dropout:** For the device selection schemes proposed in [151], [302], [303], the authors assumed that the wireless connection of each device was always available. Nevertheless, in practical wireless systems, some devices may become inactive due to poor connectivity and energy constraints, namely, device dropout. Thus, they may leave the FL process and cannot participate in model aggregation, which can severely degrade the performance of FL, such as low learning accuracy and low convergence speed [304]. To deal with this issue, new FL algorithms need to be designed to be robust for the network, where only a small number of dynamic users exist for model aggregation [305]. Also, through designing the communication protocol, devices can actively deliver local models to the edge server when they are in good connectivity conditions.

- **Hierarchical FL:** Accuracy and latency are two main factors for FL over wireless networks. The dynamic wireless environment can severely affect the transmission performance, which can lead to a low transmission rate and high transmission errors, and further lead to high transmission latency and low learning accuracy, especially for hierarchical FL (HFL). HFL is an architecture that deploys FL in heterogeneous wireless networks with three levels, including devices, SBS, and MBS [306]. In each time slot, a set of devices are selected to train the allocated global model using FedSGD algorithms [307], and then the local models are transmitted to their corresponding SBSs to be aggregated. The MBS and SBSs communicate with each other periodically to maintain a central model. HFL combines the advantages of edge FL and cloud FL. The cloud server can access more learning models, and the edge server enjoys more efficient communications with devices, leveraging edge servers as intermediaries to perform partial model aggregation in proximity, and relieve core network transmission overhead [308], [309]. However, the disadvantages of HFL are that the cloud may have excessive communication overhead and high latency, especially when a large number of devices and edge servers exist in wireless networks. One possible solution is to consider the joint design of device clustering, asynchronous FL, and communication efficiency, and use DRL algorithms to select optimal edge servers and devices for model aggregation in different time slots [310].
- **Cooperative Edge Computing for FL:** When performing FL in a network with multiple edge servers, if there is only a small number of mobile devices exist, only one edge server can be selected for model aggregation, which can save computation resources and energy for other edge servers. If the mobile device is far away from the active edge server, the learning model downloading/uploading can be done in Device-to-Device (D2D) connections. However, when a large number of mobile devices exist, selecting only one edge server for model aggregation can lead to high overhead and workload. Thus, multiple edge servers should be active for model aggregation. In this case, mobile devices need to select the optimal edge servers for model aggregation, and active edge servers should further select one edge server for global model aggregation.
- **Fully Decentralized FL:** Fully decentralized FL is usually used in a scenario with no servers or there is a failure or an attack on the server. Model weights are transmitted by D2D communication. There are no research works considering how wireless factors, such as transmission power, wireless channel, and spectrum resource allocation, affect the convergence rate and learning accuracy of fully decentralized FL. Meanwhile, because of limited communication resources in D2D transmission, how to select proper devices for model sharing still needs to be investigated.

B. Meta Learning

1) Meta Learning Methodologies:

- **Non-stationary Data Distribution:** When a new task does not exist in the experience buffer, meta learning cannot guarantee learning convergence and accuracy [311]. Meta learning also requires a large number of task datasets. Usually, authors assume that tasks are independently and identically distributed [312], and do not consider non-stationary distribution. If the task datasets change dynamically and do not have the same distribution, meta learning cannot adapt to the variation efficiently. Thus, it is difficult for meta learning to address complex datasets. One potential way to solve this issue is to propose a meta learning algorithm that is robust to tasks with a non-stationary distribution.
- **Robustness for Meta Learning:** Most meta learning algorithms are trained and tested using a small number of benchmark datasets, which means that the characteristics of datasets used for training are close to the datasets for testing. Thus, to the best of our knowledge, there are no meta-learning frameworks that not only can quickly adapt to new tasks with the help of prior experience, but also are robust to bad data samples, e.g. mislabeled data or outliers.
- **High Computation Costs:** Meta learning has the ability to wide the applicability of deep learning algorithms to more real-world domains. Consequently, increasing the generalization ability of meta learning algorithms is quite important. Although meta learning can learn new tasks quickly, meta training can be quite computationally expensive. Therefore, how to decrease the required computation latency and memory costs of meta learning remains an open challenge.
- **Theoretical Convergence Analysis:** Except for the theoretical convergence analysis of MAML, the convergence theory of other metric-based, model-based, and gradient-based meta learning algorithms is not derived. Therefore, detailed convergence analysis of meta learning is need still to be investigated.

2) Meta Learning over Wireless Networks:

- **Theoretical Analysis of Wireless Factors in Meta Learning:** It is possible that meta learning involves training a large number of meta learners, which requires much more communication demand than traditional learning approaches, especially when the number of tasks or meta learners increases exponentially. Thus, factors of the dynamic wireless environment, such as channel state information, transmission error, transmission energy, and computation capability of each device, can affect the performance of meta learning, and how these factors affect the performance of meta learning should be studied.
- **Efficient Task Training:** Plenty of tasks may exist in wireless networks corresponding to a large number of devices. Thus, how to effectively sample tasks to train meta learning to satisfy the requirements of all devices and adapt to new tasks from the dynamic environment are of utmost importance.

C. Federated Meta Learning

1) FedMeta Methodologies:

- **Privacy for FedMeta:** In current FL, the shared global model still includes all devices' privacy implicitly, while in FedMeta, a meta-learner is shared. Thus, whether FedMeta has additional advantages in protecting device privacy from the model attack perspective [313]–[316] still needs to be explored.
- **Efficient FedMeta:** Incorporating the experience replay and parameter isolation approaches into the proposed ADMM-FedMeta in [258] may further mitigate the catastrophic forgetting. In addition, although ADMM-FedMeta can be directly applied to reinforcement learning, it may result in low sample efficiency. Thus, it is essential to develop efficient collaborative reinforcement learning for FedMeta.

2) FedMeta over Wireless Networks:

- **Multi-Model FedMeta:** When a large number of tasks and devices exist in wireless networks, the overhead can be extremely large, which severely affects the transmission quality, increases the transmission latency, and may decrease the learning accuracy. To deal with these issues, one potential solution is to consider multi-model FedMeta, where devices are clustered into multiple groups, aggregate meta models in advance, and transmit them to the server for further aggregation. Thus, the possibility of multi-model FedMeta needs to be further investigated.
- **Theoretical Analysis of Wireless Factors in FedMeta:** Also, with a large number of tasks and devices existing in wireless networks, how to characterize the convergence properties and communication complexity of FedMeta considering factors of wireless networks, such as channel state information and transmission error, require further study.

XI. Conclusions

In this paper, we presented a comprehensive tutorial on the research evolution on FL, meta-learning, and FedMeta methodologies and their applications over wireless networks. We introduced the design, optimization, and evolution of these three learning approaches, providing a detailed literature review and identifying future research opportunities. By examining the advancements and challenges in FL, meta-learning, and FedMeta, we aimed to provide valuable insights and guidelines for optimizing, designing, and operating these learning algorithms in future methodologies and wireless networks, particularly in the context of emerging 6G networks.

References

[1] K. Lueth, "State of the IoT 2018: Number of IoT devices now at 7b- market accelerating," *https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/*, Aug. 2019.

[2] Cisco, "Cisco annual internet report (2018–2023) white paper," Mar. 2020.

[3] L. Andre, "53 important statistics about how much data is created every day," *https://financesonline.com/how-much-data-is-created-every-day/.*, Jan, 2023.

[4] H. Zhou, C. She, Y. Deng, M. Dohler, and A. Nallanathan, "Machine learning for massive industrial internet of things," *IEEE Wireless Commun.*, vol. 28, no. 4, pp. 81 – 87, Aug. 2021.

[5] N. Jiang, Y. Deng, and A. Nallanathan, "Traffic prediction and random access control optimization: Learning and non-learning-based approaches," *IEEE Commun. Mag.*, vol. 59, no. 3, pp. 16 – 22, Mar. 2021.

[6] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyan, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *arxiv:1604.07316*, 2016.

[7] C. M. Bishop, "Pattern recognition and machine learning," *Springer*, 2006.

[8] L. Jin, Y. Chen, T. Wang, P. Hui, and A. V. Vasilakos, "Understanding user behavior in online social networks: a survey," *IEEE Commun. Mag.*, vol. 51, no. 9, pp. 144 – 150, Sept. 2013.

[9] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," *2009 Fifth International Joint Conference on INC, IMS and IDC*, pp. 44 – 51, Aug. 2009.

[10] P. Li, J. Li, Z. Huang, T. Li, C. Gao, S. Yiu, and K. Chen, "Multi-key privacy-preserving deep learning in cloud computing," *Future Gener. Comput. Syst.*, vol. 74, pp. 76 – 85, Sept. 2017.

[11] X. Liu and Y. Deng, "Learning-based prediction, rendering and association optimization for mec-enabled wireless virtual reality (vr) networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 10, pp. 6356 – 6370, Oct. 2021.

[12] X. Liu, X. Li, and Y. Deng, "Learning-based prediction and proactive uplink retransmission for wireless virtual reality network," *IEEE Trans. Veh. Technol.*, vol. 70, no. 10, pp. 10 723 – 10 734, Oct. 2021.

[13] X. Liu, Y. Deng, C. Han, and M. D. Renzo, "Learning-based prediction, rendering and transmission for interactive virtual reality in ris-assisted terahertz networks," *IEEE J. Sel. Areas Commun.*, vol. Early Access, pp. 1 – 1, Oct. 2021.

[14] G. Ananthanarayanan, P. Bahl, P. Bodik, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58 – 67, Oct. 2017.

[15] G. Tsoumakas and I. Vlahavas, "Distributed data mining," *Encyclopedia of Data Warehousing and Mining*, 2009.

[16] D. Liu, G. Zhu, Q. Zeng, J. Zhang, and K. Huang, "Wireless data acquisition for edge learning: Data-importance aware retransmission," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 406 – 420, Jan. 2021.

[17] B. Custers, A. Sears, F. Dechesne, I. Georgieva, T. Tani, and S. van der Hof, "Eu personal data protection in policy and practice," *Hague, The Netherlands: TMC Asser Press*, 2019.

[18] B. M. Gaff, H. E. Sussman, and J. Geetter, "Privacy and big data," *Computer*, vol. 47, no. 6, pp. 7 – 9, 2014.

[19] A. Galakatos, A. Crotty, and T. Kraska, "Distributed machine learning," *Encyclopedia of Database Systems*, pp. 1196 – 1201, 2018.

[20] M. Asad, A. Moustafa, and T. Ito, "Federated learning versus classical machine learning: A convergence comparison," *arxiv:2107.10976*, 2021.

[21] L. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 10, pp. 993 – 1001, Oct. 1990.

[22] M. Wang, W. Fu, X. He, S. Hao, and X. Wu, "A survey on large-scale machine learning," *IEEE Trans. Knowl. Data Eng.*, 2020.

[23] N. Mucke, E. Reiss, J. Rungenhagen, and M. Klein, "Data splitting improves statistical performance in overparametrized regimes," *arxiv:2110.10956*, 2021.

[24] E. Dellis, B. Seeger, and A. Vlachou, "Nearest neighbor search on vertically partitioned high-dimensional data," *in Proc. 7th Int. Conf. Data Ware. and Knowl. Discov.*, pp. 243 – 253, Aug. 2005.

[25] O. L. Mangasarian, E. W. Wild, and G. M. Fung, "Privacy-preserving classification of vertically partitioned data via random kernels," *ACM Trans. Knowl. Discov. Data*, vol. 2, no. 3, pp. 1 – 16, 2008.

[26] J. Vaidya, H. Yu, and X. Jian, "Privacy-preserving svm classification," *Knowl. Inf. Syst.*, vol. 14, no. 2, pp. 161 – 178, 2008.

[27] Z. Yang and R. N. Wright, "Improved privacy-preserving bayesian network parameter learning on vertically partitioned data," *in Proc. 21st Int. Conf. Data Eng. Workshops (ICDEW)*, 2005.

[28] Z. Yang, "Privacy-preserving computation of bayesian networks on vertically partitioned data," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 9, pp. 1253 – 1264, 2006.

[29] W. Fang and B. Yang, "Privacy preserving decision tree learning over vertically partitioned data," *in Proc. 2008 Int. Conf. Computer Science and Software Engineering*, pp. 1049 – 1052, 2008.

[30] J. Vaidya, C. Clifton, M. Kantarcioglu, and A. S. Patterso, "Privacy-preserving decision trees over vertically partitioned data," *ACM Trans. Knowl. Discov. Data*, vol. 2, no. 3, pp. 1 – 27, 2008.

[31] H. Zheng, S. R. Kulkarni, and H. V. Poor, "Attribute-distributed learning: Models, limits, and algorithms," *IEEE Trans. Signal Process.*, vol. 59, no. 1, pp. 386 – 398, Jan. 2021.

[32] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," *In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1273 – 1282, Feb. 2017.

[33] K. Sozinov, V. Vlassov, and S. Girdzijauskas, "Human activity recognition using federated learning," *in Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl. Ubiquitous Comput. Commun. Big Data Cloud Comput. Soc. Comput. Netw. Sustain. Comput. Commun*, pp. 1103 – 1111, 2018.

[34] T. Yu, T. Li, Y. Sun, S. Nanda, V. Smith, V. Sekar, and S. Seshan, "Learning context-aware policies from multiple smart homes via federated multi-task learning," *in Proc. IEEE/ACM Int. Conf. Internet Things Design Implement. (IoTDI)*, pp. 104 – 115, 2020.

[35] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," *Int. J. Med. Informat.*, vol. 112, pp. 59 – 67, Apr. 2018.

[36] K. Powell, "Nvidia clara federated learning to deliver ai to hospitals while protecting patient data," 2019.

[37] D. Verma, S. Julier, and G. Cirincione, "Federated ai for building ai solutions across multiple agencies," *arxiv:1809.10036*, 2018.

[38] M. Arambakam and J. Beel, "Federated meta-learning: Democratizing algorithm selection acros disciplines and software libraries," *In ICML Workshop*, 2020.

[39] S. Trindade, L. F. Bittencourt, and N. L. da Fonseca, "Resource management at the network edge for federated learning," *Digit. Commun. Netw.*, pp. 2352–8648, 2022.

[40] N. Schweighofer and K. Doya, "Meta-learning in reinforcement learning," *Neural. Netw.*, vol. 16, no. 1, pp. 5 – 9, 2003.

[41] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1 – 19, Feb. 2019.

[42] Y. Jin, X. Wei, Y. Liu, and Q. Yang, "Towards utilizing unlabeled data in federated learning: A survey and prospective," *arxiv:2002.11545*, 2020.

[43] P. M. Mammen, "Federated learning: Opportunities and challenges," *arxiv:2101.05428*, 2021.

[44] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowl. Based Syst.*, vol. 216, Mar. 2021.

[45] S. Ji, T. Saravirta, S. Pan, G. Long, and A. Walid, "Emerging trends in federated learning: From model fusion to federated X learning," *arxiv:2102.12920*, 2021.

[46] O. Shahid, S. Pouriyeh, R. M. Parizi, Q. Z. Sheng, G. Srivastava, and L. Zhao, "Communication efficiency in federated learning: Achievements and challenges," *arxiv:2107.10996*, 2021.

[47] Y. Shi, H. Yu, and C. Leung, "A survey of fairness-aware federated learning," *arxiv:2111.01872*, 2021.

[48] J. Qi, Q. Zhou, L. Lei, and K. Zheng, "Federated reinforcement learning: Techniques, applications, and open challenges," *arxiv:2108.11887*, 2021.

[49] M. Liu, S. Ho, M. wang, L. Gao, Y. Jin, and H. Zhang, "Federated learning meets natural language processing: A survey," *arxiv:2107.12603*, 2021.

[50] R. Zeng, C. Zeng, X. Wang, B. Li, and X. Chu, "A comprehensive survey of incentive mechanism for federated learning," *arxiv:2106.15406*, 2021.

[51] Y. Zhan, J. Zhang, Z. Hong, L. Wu, P. Li, and S. Guo, "A survey of incentive mechanism design for federated learning," *IEEE Trans. Emerg. Topics Comput.*, vol. Early Access, pp. 1 – 1, Mar. 2021.

[52] O. J. Ciceri, C. A. Astudillo, Z. Zhu, and N. L. S. da Fonseca, "Federated learning over next-generation ethernet passive optical networks," *IEEE Netw.*, pp. 1–9, 2022.

[53] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless network intelligence at the edge," *Proceedings of the IEEE*, vol. 107, no. 11, pp. 2204 – 2239, Nov. 2019.

[54] S. Niknam, H. S. Dhillon, and J. H. Reed, "Federated learning for wireless communications: Motivation, opportunities, and challenges," *IEEE Commun. Mag.*, vol. 58, no. 6, pp. 46 – 51, Jun. 2020.

[55] Z. Zhao, C. Feng, H. H. Yang, and X. Luo, "Federated-learning-enabled intelligent fog radio access networks: Fundamental theory, key techniques, and future trends," *IEEE Wireless Commun.*, vol. 27, no. 2, pp. 22 – 28, Apr. 2020.

[56] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Toward an intelligent edge: Wireless communication meets machine learning," *IEEE Commun. Mag.*, vol. 58, no. 1, pp. 19 – 25, Jan. 2020.

[57] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50 – 60, May 2020.

[58] K. Yang, Y. Shi, Y. Zhou, Z. Yang, L. Fu, and W. Chen, "Federated machine learning for intelligent IoT via reconfigurable intelligent surface," *IEEE Netw.*, vol. 34, no. 5, pp. 16 – 22, Sept. 2020.

[59] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y. C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031 – 2063, 3rd Quart. 2020.

[60] M. Chen, H. V. Poor, W. Saad, and S. Cui, "Wireless communications for collaborative federated learning," *IEEE Commun. Mag.*, vol. 58, no. 12, pp. 48 – 54, Dec. 2020.

[61] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A survey on federated learning for resource-constrained IoT devices," *IEEE Internet of Things J.*, vol. Early Access, pp. 1 – 1, Jul. 2021.

[62] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for internet of things: A comprehensive survey," *arxiv:2104.07914*, 2021.

[63] J. Park, S. Samarakoon, A. Elgabli, J. Kim, M. Bennis, S. L. Kim, and M. Debbah, "Communication-efficient and distributed learning over wireless networks: Principles and applications," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 796 – 819, May 2021.

[64] M. Chen, D. Gunduz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3579 – 3605, Dec. 2021.

[65] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous federated learning on heterogeneous devices: A survey," *arxiv:2109.04269*, 2021.

[66] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artif. Intell. Rev.*, vol. 18, pp. 77 – 95, 2002.

[67] C. Lemke, M. Budka, and B. Gabrys, "Metalearning: a survey of trends and technologies," *Artif. Intell. Rev.*, vol. 44, pp. 117 – 130, 2015.

[68] J. Vanschoren, "Meta-learning: A survey," *arxiv:1810.03548*, 2018.

[69] W. Yin, "Meta-learning for few-shot natural language processing: A survey," *arxiv:2007.09604*, 2020.

[70] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *arxiv:2004.05439*, 2020.

[71] H. Peng, "A comprehensive overview and survey of recent advances in meta-learning," *arxiv:2004.11149*, 2020.

[72] Y. Ma, S. Zhao, W. Wang, Y. Li, and I. King, "Multimodality in meta-learning: A comprehensive survey," *arxiv:2109.13576*, 2021.

[73] E. P. Xing, Q. Ho, P. Xie, and D. Wei, "Strategies and principles of distributed machine learning on big data," *Engineering*, vol. 2, no. 2, pp. 179 – 195, Jun. 2016.

[74] S. C. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," *arxiv:1802.02871*, 2018.

[75] R. F. Prudencio, M. R. O. A. Maximo, and E. L. Colombini, "A survey on offline reinforcement learning: Taxonomy, review, and open problems," *arxiv:2203.01387*, 2022.

[76] A. A. Benczur, L. Kocsis, and R. Palovics, "Online machine learning in big data streams," *arxiv:1802.05872*, 2018.

[77] P. Baran, "On distributed communications networks," *IEEE Trans. Commun. Syst.*, vol. 12, no. 1, pp. 1 – 9, Mar. 1964.

[78] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *arxiv:1912.09789*, 2019.

[79] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, "A reliable effective terascale linear learning system," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1111 – 1133, 2014.

[80] J. Wei, W. Dai, A. Qiao, Q. Ho, H. Cui, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Managed communication and consistency for fast data-parallel iterative analytics," *In SoCC*, 2015.

[81] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, "Parameter server for distributed machine learning," *in Proc. Big Learning NIPS Workshop*, 2013.

[82] J. Wei, W. Dai, A. Qiao, Q. Ho, H. Cui, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Managed communication and consistency for fast data-parallel iterative analytics," *In SoCC*, pp. 381 – 394, 2015.

[83] I. Foster and A. Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing," *in Proceedings of the International Workshop on Peer-to-Peer Systems. Springer*, pp. 118 – 128, 2003.

[84] A. Elgabli, J. Park, A. S. Bedi, M. Bennis, and V. Aggarwal, "Gadmm: Fast and communication efficient framework for distributed machine learning," *J. Mach. Learn. Res.*, vol. 21, no. 76, pp. 1 – 39, 2020.

[85] A. Elgabli, J. Park, A. S. Bedi, C. B. Issaid, M. Bennis, and V. Aggarwal, "Q-gadmm: Quantized group admm for communication efficient decentralized machine learning," *IEEE Trans. Commun.*, vol. 69, no. 1, pp. 164 – 181, Jan. 2021.

[86] C. B. Issaid, A. Elgabli, J. Park, M. Bennis, and M. Debbah, "Communication efficient decentralized learning over bipartite graphs," *IEEE Trans. Commun.*, vol. 21, no. 6, pp. 4150 – 4167, Jun. 2022.

[87] M. G. Kibria, G. P. Villardi, K. Nguyen, K. Ishizu, and F. Kojima, "Heterogeneous networks in shared spectrum access communications," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 1, pp. 145 – 158, Jan. 2017.

[88] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless network intelligence at the edge," *arxiv:1812.02858*, 2018.

[89] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," *in Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, pp. 1322 – 1333, Oct. 2015.

[90] N. Barroso, E. Camara, M. V. Luzon, G. G. Seco, M. A. Veganzones, and F. Herrera, "Dynamic federated learning model for identifying adversarial clients," *arxiv:2007.15030*, 2020.

[91] M. Mohri, A. Rostamizadeh, and A. Talwalkar, "Foundations of machine learning," *MIT press*, 2012.

[92] S. J. Russell and P. Norvig, "Artificial intelligence: A modern approach," *Prentice Hall*, 2010.

[93] A. Tharwat, "Principal component analysis - a tutorial," *Pattern Recognit.*, vol. 3, no. 3, pp. 197 – 240, 2016.

[94] G. Hinton and T. J. Sejnowski, "Unsupervised learning: foundations of neural computation," *MIT press*, 1999.

[95] N. Lu, Z. Wang, X. Li, G. Niu, Q. Dou, and M. Sugiyama, "Federated learning from only unlabeled data with class-conditional-sharing clients," *in ICLR*, 2022.

[96] A. Grammenos, R. Mendoza Smith, J. Crowcroft, and C. Mascolo, "Federated principal component analysis," in *Adv. Neural Inf. Process. Syst*, vol. 33, 2020, pp. 6453–6464.

[97] Y. Cheung, J. Jiang, F. Yu, and J. Lou, "Vertical federated principal component analysis and its kernel extension on feature-wise distributed data," *arxiv:2203.01752*, 2022.

[98] X. Zhu, "Semi-supervised learning literature survey," *Technical report, University of Wisconsin-Madison Department of Computer Sciences*, 2005.

[99] H. Lin, J. Lou, L. Xiong, and C. Shahabi, "Semifed: Semi-supervised federated learning with consistency and pseudo-labeling," *arxiv:2108.09412*, 2021.

[100] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 14 413 – 14 423, Dec. 2020.

[101] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237 – 285, Jan. 1996.

[102] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, "A comprehensive survey of loss functions in machine learning," *Annals of Data Science*, Apr. 2020.

[103] H. B. McMahan, E. Moore, D. Ramage, and B. A. Arcas, "Federated learning of deep networks using model averaging," *CoRR abs/1602.05629*, 2016.

[104] A. Sapio, M. Canini, C. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," *arxiv:1903.06701*, 2019.

[105] J. Kittler, "Combining classifiers: a theoretical framework," *Pattern Anal. Appl.*, pp. 18 – 27, 1998.

[106] T. K. Ho, J. J. Hull, and S. N. Srihari, "Decision combination in multiple classifier systems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 1, pp. 66 – 75, Jan. 1994.

[107] S. Ek, F. Portet, P. Lalanda, and G. Vega, "Evaluation of federated learning aggregation algorithms application to human activity recognition," *UbiComp/ISWC '20*, pp. 638 – 643, Sep. 2020.

[108] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE TNNLS*, 2018.

[109] Y. Yeganeh, A. Farshad, N. Navab, and S. Albarqouni, "Inverse distance aggregation for federated learning with non-iid data," *arxiv:2008.07665*, 2020.

[110] S. Ji, S. Pan, G. Long, X. Li, J. Jiang, and Z. Huang, "Learning private neural language modeling with attentive aggregation," *arxiv:1812.07108*, 2018.

[111] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshal, "Activation functions: Comparison of trends in practice and research for deep learning," *arxiv:1811.03378*, 2018.

[112] J. Zhang, S. P. Karimireddy, A. Veit, S. Kim, S. J. Reddi, S. Kumar, and S. Sra, "Why ADAM beats SGD for attention models," *arxiv:1912.03194*, 2019.

[113] E. Gorbunov, M. Danilova, and A. Gasnikov, "Stochastic optimization with heavy-tailed noise via accelerated gradient clipping," *arxiv:2005.10785*, 2020.

[114] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," *arxiv:1910.06378*, 2019.

[115] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," *MIT press*, 2016.

[116] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konecny, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arxiv:2003.00295*, 2020.

[117] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arxiv:1904.09237*, 2019.

[118] M. Zaheer, D. S. S. S. Reddi, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," *Advances in Neural Information Processing Systems*, 2018.

[119] C. Barakat, E. Altman, and W. Dabbous, "On TCP performance in a heterogeneous network: a survey," *IEEE Commun. Mag.*, vol. 38, no. 1, pp. 40 – 46, Jan. 2000.

[120] D. Cavalcanti, D. Agrawal, C. Cordeiro, B. Xie, and A. Kumar, "Issues in integrating cellular networks WLANs, AND MANETs: a futuristic heterogeneous wireless network," *IEEE Wireless Commun.*, vol. 12, no. 3, pp. 30 – 41, Jun. 2005.

[121] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," *arxiv:1902.01046*, 2019.

[122] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *arxiv:1812.06127*, 2018.

[123] H. T. Nguyen, V. Sehwag, S. Hosseinalipour, C. G. Brinton, M. Chiang, and H. V. Poor, "Fast-convergent federated learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 201 – 218, Jan. 2021.

[124] X. Wu, R. Guo, D. Simcha, D. Dopson, and S. Kumar, "Efficient inner product approximation in hybrid spaces," *arxiv:1903.08690*, 2019.

[125] J. Konecny, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arxiv:1610.05492*, 2016.

[126] A. T. Suresh, F. X. Yu, S. Kumar, and H. B. McMahan, "Distributed mean estimation with limited communication," *In ICML*, pp. 3329 – 3337, 2017.

[127] H. Zhu and Y. Jin, "Multi-objective evolutionary federated learning," *arxiv:1812.07478*, 2018.

[128] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature Commun.*, vol. 9, no. 1, 2018.

[129] R. Jin, Y. Huang, X. He, H. Dai, and T. Wu, "Stochastic-sign sgd for federated learning with theoretical guarantees," *arxiv:2002.10940*, 2020.

[130] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S. Kim, "Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data," *arxiv:1811.11479*, 2018.

[131] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *NIPS Deep Learning Workshop*, Dec. 2014.

[132] W. Huang, T. Li, D. Wang, S. Du, and J. Zhang, "Fairness and accuracy in federated learning," *arxiv:2012.10069*, 2020.

[133] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," *In ICML*, 2019.

[134] T. Li, M. Sanjabi, A. Beirami, and V. Smith, "Fair resource allocation in federated learning," *arxiv:1905.10497*, 2019.

[135] X. Zhou, "On the fenchel duality between strong convexity and lipschitz continuous gradient," *arxiv:1803.06573*, 2018.

[136] L. Lyu, X. Xu, and Q. Wang, "Collaborative fairness in federated learning," *arxiv:2008.12161*, 2020.

[137] EU, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)," *Official Journal of the European Union*, pp. 1 – 88, May 2016.

[138] M. Yurochkin, M. Agarwal, S. Ghosh, K. Greenewald, N. Hoang, and Y. Khazaeni, "Bayesian nonparametric federated learning of neural networks," *In ICML*, 2019.

[139] R. Thibaux and M. I. Jordan, "Hierarchical beta processes and the indian buffet process," *Artificial intelligence and statistics*, pp. 564 – 571, 2007.

[140] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *arxiv:2002.06440*, 2020.

[141] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *arxiv:2006.04088*, 2020.

[142] C. Briggs, Z. Fan, and P. Andras, "Federated learning with hierarchical clustering of local updates to improve training on non-iid data," *arxiv:2004.11791*, 2020.

[143] M. Xie, G. Long, T. Shen, T. Zhou, X. Wang, and J. Jiang, "Multi-center federated learning," *arxiv:2005.01026*, 2020.

[144] H. Kavalionak, E. Carlini, P. Dazzi, L. Ferrucci, M. Mordacchini, and M. Coppola, "Impact of network topology on the convergence of decentralized federated learning systems," *in ISCC*, 2021.

[145] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Third workshop on bayesian deep learning (NeurIPS)*, vol. 2, 2018.

[146] P. L. Bartlett, "The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network," *IEEE Trans. Inf. Theory*, vol. 44, no. 2, pp. 525 – 536, Mar. 1998.

[147] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," *in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS'16)*, pp. 901 – 909, Dec. 2016.

[148] H. H. Yang, Z. Liu, T. Q. S. Quek, and H. V. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE Trans. Commun.*, vol. 68, no. 1, pp. 317 – 333, Jan. 2020.

[149] J. G. Choi and S. Bahk, "Cell-throughput analysis of the proportional fair scheduler in the single-cell environment," *IEEE Trans. Veh. Technol.*, vol. 56, no. 2, pp. 766 – 778, Mar. 2007.

[150] J. Ren, Y. He, D. Wen, G. Yu, K. Huang, and D. Guo, "Scheduling for cellular federated edge learning with importance and channel awareness," *IEEE Trans. Wireless Commun.*, vol. 19, no. 11, pp. 7690 – 7703, Nov. 2020.

[151] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," *arxiv:1804.08333*, 2018.

[152] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 269 – 283, Jan. 2021.

[153] R. Jonker and T. Volgenant, "Improving the hungarian assignment algorithm," *Oper. Res. Lett.*, vol. 5, no. 4, pp. 171 – 175, Oct. 1986.

[154] M. S. H. Abad, E. Ozfatura, D. Gunduz, and O. Ercetin, "Hierarchical federated learning across heterogeneous cellular networks," *arxiv:1909.02362*, 2019.

[155] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *in ICLR*, 2018.

[156] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. S. Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1935 – 1949, Mar. 2021.

[157] J. Ren, G. Yu, and G. Ding, "Accelerating DNN training in wireless federated edge learning systems," *arxiv:1905.09712*, 2019.

[158] Z. Tao and Q. Li, "Esgd: Communication efficient distributed deep learning on the edge," *in Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, pp. 1 – 6, 2018.

[159] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," *in NIPS*, pp. 2595 – 2603, 2010.

[160] S. Ruder, "An overview of gradient descent optimization algorithms," *arxiv:1609.04747*, 2016.

[161] N. H. Tran, W. Bao, A. Zomaya, N. M. Nh, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," *in Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pp. 1387 – 1395, Apr. 2019.

[162] N. Hashimzade, G. Myles, and J. Black, "A dictionary of economics, 5th ed." *Oxford University Press*, 2017.

[163] M. M. Wadu, S. Samarakoon, and M. Bennis, "Federated learning under channel uncertainty: Joint client scheduling and resource allocation," *arxiv:2002.00802*, 2020.

[164] M. Chen, H. V. Poor, W. Saad, and S. Cui, "Convergence time optimization for federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 4, pp. 2457 – 2471, Apr. 2021.

[165] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," *in International Conference on Learning Representations Workshop*, 2016.

[166] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *in NeurIPS workshop on Optimization for Machine Learning*, 2020.

[167] Z. C. C. H. Hu and E. G. Larsson, "Scheduling and aggregation design for asynchronous federated learning over wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 874 – 886, Apr. 2023.

[168] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-iid data," *arxiv:1911.06268*, 2019.

[169] J. Liu, H. Xu, L. Wang, Y. Xu, C. Qian, J. Huang, and H. Huang, "Adaptive asynchronous federated learning in resource-constrained edge computing," *IEEE Trans. Mob. Comput.*, pp. 1 – 1, Dec 2020.

[170] X. Zhou, Y. Deng, H. Xia, S. Wu, and M. Bennis, "Time-triggered federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. Early Access, pp. 1 – 1, Jul. 2022.

[171] W. Liu, X. Zang, Y. Li, and B. Vucetic, "Over-the-air computation systems: Optimization, analysis and scaling laws," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5488 – 5502, Aug. 2020.

[172] W. Fang, Y. Jiang, Y. Shi, Y. Zhou, W. Chen, and K. B. Letaief, "Over-the-air computation via reconfigurable intelligent surface," *IEEE Trans. Commun.*, vol. 69, no. 12, pp. 8612 – 8626, Dec. 2021.

[173] M. Goldenbaum, H. Boche, and S. Stanczak, "Harnessing interference for analog function computation in wireless sensor networks," *IEEE Trans. Signal Process.*, vol. 61, pp. 4893 – 4906, Oct. 2013.

[174] K. Yang, T. Jiang, Y. Shi, and Z. Ding, "Federated learning via over-the-air computation," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 2022 – 2035, Mar. 2020.

[175] L. Chen, X. Qin, and G. Wei, "A uniform-forcing transceiver design for over-the-air function computation," *IEEE Wireless Commun. Lett.*, vol. 7, pp. 942 – 945, Dec. 2018.

[176] G. Zhu, Y. Wang, and K. Huang, "Broadband analog aggregation for low-latency federated edge learning," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 491 – 506, Jan. 2020.

[177] G. Zhu, Y. Du, D. Gündüz, and K. Huang, "One-bit over-the-air aggregation for communication-efficient federated edge learning: Design and convergence analysis," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 2120 – 2135, Nov. 2021.

[178] C. Zhong, H. Yang, and X. Yuan, "Over-the-air federated multi-task learning over MIMO multiple access channels," *IEEE Trans. Wireless Commun.*, vol. 22, no. 6, pp. 3853–3868, Jun. 2023.

[179] N. Zhang and M. Tao, "Gradient statistics aware power control for over-the-air federated learning," *IEEE Trans. Wireless Commun.*, vol. 20, no. 8, pp. 5115 – 5128, Aug. 2021.

[180] Y. Xue, L. Su, and V. K. N. Lau, "Fedocomp: Two-timescale online gradient compression for over-the-air federated learning," *IEEE Internet Things J.*, vol. 9, no. 19, pp. 19 330 – 19 345, Oct. 2022.

[181] M. H. Adeli and A. Sahin, "Multi-cell non-coherent over-the-air computation for federated edge learning," *in Proc. IEEE International Conference on Communications (ICC)*, pp. 1 – 6, Apr. 2022.

[182] Z. Wang, Y. Zhao, Y. Zhou, Y. Shi, C. Jiang, and K. B. Letaief, "Over-the-air computation: Foundations, technologies, and applications," *arxiv:2210.10524*, 2022.

[183] X. Wei, C. Shen, J. Yang, and H. V. Poor, "Random orthogonalization for federated learning in massive mimo systems," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 3382–3387.

[184] Y. Shao, D. Gündüz, and S. C. Liew, "Federated edge learning with misaligned over-the-air computation," *IEEE Trans. Wireless Commun.*, vol. 21, no. 6, pp. 3951–3964, Jun. 2022.

[185] A. Şahin, B. Everette, and S. S. Muhtasimul Hoque, "Distributed learning over a wireless network with fsk-based majority vote," in *Proc. Int. Conf. Adv. Commun. Technol. Netw. (CommNet)*, Dec. 2021.

[186] A. Şahin, B. Everette, and S. S. M. Hoque, "Over-the-air computation with dft-spread ofdm for federated edge learning," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Dec. 2022.

[187] A. Sahin, "Over-the-air computation based on balanced number systems for federated edge learning," *arxiv:2210.07012*, 2022.

[188] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," *In ICML Deep Learning Workshop*, 2015.

[189] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," *In NeurIPS*, 2016.

[190] "Cosine distance, cosine similarity, angular cosine distance, angular cosine similarity," *www.itl.nist.gov*, 2020.

[191] J. Snell, K. Swersky, and R. Z. Zemel, "Prototypical networks for few-shot learning," *In NeurIPS*, 2017.

[192] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," *In CVPR*, 2018.

[193] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," *In ICML*, 2016.

[194] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arxiv:1410.5401*, 2014.

[195] T. Munkhdalai and H. Yu, "Meta networks," *In ICML*, 2017.

[196] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl2: Fast reinforcement learning via slow reinforcement learning," *arxiv:1611.02779*, 2016.

[197] J. X. Wang, Z. K. Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arxiv:1611.05763*, 2016.

[198] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," *in ICLR*, 2018.

[199] A. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arxiv:1609.03499*, 2016.

[200] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arxiv:1706.03762*, 2017.

[201] L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson, "Fast context adaptation via meta-learning," *in ICML*, 2019.

[202] K. Ji, J. D. Lee, Y. Liang, and H. V. Poor, "Convergence of meta-learning with task-specific adaptation over partial parameters," *in NIPS*, 2020.

[203] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *In ICML*, pp. 1126 – 1135, 2017.

[204] K. Chua, Q. Lei, and J. D. Lee, "How fine-tuning allows for effective meta-learning," *arxiv:2105.02221*, 2021.

[205] Y. Hu, S. Zhang, X. Chen, and N. He, "Biased stochastic first-order methods for conditional stochastic optimization and applications in meta learning," *in NIPS*, 2020.

[206] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arxiv:1803.02999*, 2018.

[207] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *in ICLR*, 2015.

[208] A. Fallah, A. Mokhtari, and A. Ozdaglar, "On the convergence theory of gradient-based model-agnostic meta-learning algorithms," *in AISTATS*, 2020.

[209] J. Martens, "Deep learning via hessian-free optimization," *in ICML*, 2010.

[210] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," *in ICML*, 2011.

[211] R. Kiros, "Training neural networks with stochastic hessian-free optimization," *arxiv:1301.3641*, 2013.

[212] J. Martens and I. Sutskever, "Training deep and recurrent networks with hessian-free optimization," *Neural Networks: Tricks of the Trade*, pp. 479 – 535, 2012.

[213] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-SGD: Learning to learn quickly for few shot learning," *arxiv:1707.09835*, 2017.

[214] A. Nichol and J. Schulman, "Reptile: a scalable meta learning algorithm," *arxiv:1803.02999*, 2018.

[215] J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. AHN, "Bayesian model-agnostic meta-learning," *in NIPS*, 2018.

[216] Q. Liu and D. Wang, "Stein variational gradient descent: A general purpose bayesian inference algorithm," *in NIPS*, 2016.

[217] C. J. Geyer, "Practical markov chain monte carlo," *Stat. Sci.*, 1992.

[218] E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths, "Recasting gradient-based meta-learning as hierarchical bayes," *in ICLR*, 2018.

[219] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," *in ICML*, 2015.

[220] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell, "Meta-learning with latent embedding optimization," *in ICLR*, 2019.

[221] A. Rajeswaran, C. Finn, S. Kakade, and S. Levine, "Meta-learning with implicit gradients," *in NIPS*, 2019.

[222] A. Antoniou, H. Edwards, and A. Storkey, "How to train your maml," *in ICLR*, 2019.

[223] S. Flennerhag, A. A. Rusu, R. Pascanu, F. Visin, H. Yin, and R. Hadsell, "Meta-learning with warped gradient descent," *in ICLR*, 2020.

[224] G. W. Flake and B. A. Pearlmutter, "Differentiating functions of the jacobian with respect to the weights," *in NIPS*, 1999.

[225] A. Griewank, "Some bounds on the complexity of gradients, jacobians, and hessians," *Complexity in numerical optimization*, 1993.

[226] C. Finn, A. Rajeswaran, S. Kakade, and S. Levine, "Online meta-learning," *In ICML*, 2019.

[227] X. Gao, X. Li, and S. Zhang, "Online learning with non-convex losses and non-stationary regret," *in AISTATS*, 2018.

[228] J. Hannan, "Approximation to bayes risk in repeated play," *Contributions to the Theory of Games*, 1957.

[229] A. Kalai and S. Vempala, "Efcient algorithms for online decision problems," *J. Comput. Syst. Sci.*, vol. 71, no. 3, pp. 291 – 307, Oct. 2005.

[230] Q. He, A. Moayyedi, G. Dan, G. P. Koudouridis, and P. Tengkvist, "A meta-learning scheme for adaptive short-term network traffic prediction," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2271 – 2283, Oct. 2020.

[231] J. Xia and G. Deniz, "Meta-learning based beamforming design for miso downlink," *arxiv:2103.11978*, 2021.

[232] Y. Yuan, G. Zheng, K. K. Wong, B. Ottersten, and Z. Q. Luo, "Transfer learning and meta learning-based fast downlink beamforming adaptation," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1742 – 1755, Mar. 2021.

[233] J. Zhang, Y. He, Y. W. Li, C. K. Wen, and S. Jin, "Meta learning-based MIMO detectors: Design, Simulation, and Experimental Test," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1122 – 1137, Feb. 2021.

[234] X. Lee, A. Havens, G. Chowdhary, and S. Sarkar, "Learning to cope with adversarial attacks," *In ICML*, 2019.

[235] A. J. Havens, Z. Jiang, and S. Sarkar, "Online robust policy learning in the presence of unknown adversaries," *In NIPS*, 2018.

[236] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," *In ICLR*, 2018.

[237] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for interference management," *IEEE Trans. Signal Process.*, vol. 66, no. 20, pp. 5438 – 5453, Oct. 2018.

[238] W. Xia, G. Zheng, Y. Zhu, J. Zhang, J. Wang, and A. P. Petropulu, "A deep learning framework for optimization of MISO downlink beamforming," *IEEE Trans. Commun.*, vol. 68, no. 3, pp. 1866 – 1880, Mar. 2019.

[239] L. Pellaco, M. Bengtsson, and J. Jalden, "Deep unfolding of the weighted MMSE beamforming algorithm," *arxiv:2006.08448*, 2020.

[240] A. Chowdhury, G. Verma, C. Rao, A. Swami, and S. Segarra, "Unfolding WMMSE using graph neural networks for efficient power allocation," *IEEE Trans. Wireless Commun.*, vol. 20, no. 9, pp. 6004 – 6017, Sept. 2021.

[241] D. Ito, S. Takabe, and T. Wadayama, "Trainable ISTA for sparse signal recovery," *IEEE Trans. Signal Process.*, vol. 67, no. 12, pp. 3113 – 3125, Jan. 2019.

[242] H. He, C. K. Wen, S. Jin, and G. Y. Li, "Model-driven deep learning for mimo detection," *IEEE Trans. Signal Process.*, vol. 68, pp. 1702 – 1715, Feb. 2020.

[243] J. Zhang, H. He, C. K. Wen, S. Jin, and G. Y. Li, "Deep learning based on orthogonal approximate message passing for cp-free OFDM," *in Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, pp. 1 – 6, May 2019.

[244] S. Sahin, C. Poulliat, A. M. Cipriano, and M. L. Boucheret, "Doubly iterative turbo equalization: Optimization through deep unfolding," *in PIMRC*, pp. 1 – 6, Sep. 2019.

[245] T. P. Minka, "A family of algorithms for approximate bayesian inference," *Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci. MIT, Cambridge, MA, USA*, 2001.

[246] M. W. Seeger, "Expectation propagation for exponential families," *Dept. Elect. Eng. Comput. Sci., Univ. Calif., Berkeley, CA, USA, Tech. Rep.*, 2005.

[247] J. Cespedes, P. M. Olmos, M. Sánchez-Fernández, and F. Perez-Cruz, "Expectation propagation detection for high-order high-dimensional MIMO systems," *IEEE Trans. Commun.*, vol. 62, no. 8, pp. 2840 – 2849, Aug. 2014.

[248] J. Cespedes, P. M. Olmos, and M. Sánchez-Fernández, "Probabilistic MIMO symbol detection with expectation consistency approximate inference," *IEEE Trans. Veh. Technol.*, vol. 67, no. 4, pp. 3481 – 3494, Apr. 2018.

[249] S. Park, H. Jang, O. Simeone, and J. Kang, "Learning to demodulate from few pilots via offline and online meta-learning," *IEEE Trans. Signal Process.*, vol. 69, pp. 226–239, 2021.

[250] I. Nikoloska and O. Simeone, "Fast power control adaptation via meta-learning for random edge graph neural networks," in *2021 IEEE 22nd International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2021, pp. 146–150.

[251] Y. Hu, M. Chen, W. Saad, H. V. Poor, and S. Cui, "Distributed multi-agent meta learning for trajectory design in wireless drone networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 10, pp. 3177–3192, 2021.

[252] K. Xiong, Z. Zhao, W. Hong, M. Peng, and T. Q. Quek, "Few-shot learning in wireless networks: A meta-learning model-enabled scheme," in *2022 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2022, pp. 415–420.

[253] C. Finn, K. Xu, and S. Levine, "Probabilistic model-agnostic meta-learning," *arxiv:1806.02817*, 2018.

[254] T. Kim, J. Yoon, O. Dia, S. Kim, Y. Bengio, and S. Ahn, "Bayesian model-agnostic meta-learning," *in NIPS*, 2018.

[255] R. Vuorio, S. Sun, H. Hu, and J. J. Lim, "Multimodal model-agnostic meta-learning via task-aware modulation," *arxiv:1910.13616*, 2019.

[256] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, "Federated meta-learning with fast convergence and efficient communication," *arxiv:1802.07876*, 2018.

[257] S. Lin, G. Yang, and J. Zhang, "A collaborative learning framework via federated meta-learning," *In ICDCS*, 2020.

[258] S. Yue, J. Ren, J. Xin, S. Lin, and J. Zhang, "Inexact-ADMM based federated meta-learning for fast and continual edge learning," *arxiv:2012.08677*, 2020.

[259] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Netw.*, vol. 113, pp. 54 – 71, 2019.

[260] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1 – 122, 2010.

[261] M. Hong, Z. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," *SIAM J. Optim.*, vol. 26, no. 1, pp. 337 – 364, 2016.

[262] S. Magnusson, P. C. Weeraddana, M. G. Rabbat, and C. Fischione, "On the convergence of alternating direction lagrangian methods for nonconvex structured optimization problems," *IEEE Trans. Control Netw. Syst.*, vol. 3, pp. 296 – 309, 2015.

[263] F. Wang, Z. Xu, and H. Xu, "Convergence of bregman alternating direction method with multipliers for nonconvex composite problems," *arxiv:1410.8625*, 2014.

[264] Y. Wang, W. Yin, and J. Zeng, "Global convergence of admm in nonconvex nonsmooth optimization," *J. Sci. Comput.*, vol. 1, pp. 29 – 63, 2019.

[265] F. Wang, W. Cao, and Z. Xu, "Convergence of multi-block bregman admm for nonconvex composite problems," *Sci. China Inf. Sci.*, vol. 12, 2018.

[266] R. F. Barber and E. Y. Sidky, "Convergence for nonconvex ADMM with applications to CT imaging," *arxiv:2006.07278*, 2020.

[267] B. Jiang, T. Lin, S. Ma, and S. Zhang, "Structured nonconvex and nonsmooth optimization: algorithms and iteration complexity analysis," *Comput. Optim. Appl.*, vol. 1, pp. 115 – 157, 2019.

[268] A. Lanza, S. Morigi, I. Selesnick, and F. Sgallari, "Nonconvex nonsmooth optimization via convex-nonconvex majorization-minimization," *Numer. Math*, vol. 2, pp. 343 – 381, 2017.

[269] M. C. Mukkamala, P. Ochs, T. Pock, and S. Sabach, "Convex-concave backtracking for inertial bregman proximal gradient algorithms in nonconvex optimization," *SIAM J. Math. Data Sci.*, vol. 3, pp. 658 – 682, 2020.

[270] C. T. Dinh, N. H. Tran, M. N. Nguyen, C. S. Hong, W. Bao, A. Y. Zomaya, and V. Gramoli, "Federated learning over wireless networks: Convergence analysis and resource allocation," *IEEE/ACM Trans. Networking*, vol. 29, no. 1, pp. 398 – 409, 2020.

[271] P. Kairouz, "Advances and open problems in federated learning," *arxiv:1912.04977*, 2019.

[272] S. Yue, J. Ren, J. Xin, D. Zhang, Y. Zhang, and W. Zhuang, "Efficient federated meta-learning over multi-access wireless networks," *arxiv:2108.06453*, 2021.

[273] S. Yue, "Efficient federated meta-learning over multi-access wireless networks (technical report)," *https://github.com/shaunyue/ural*, 2021.

[274] A. Elgabli, C. B. Issaid, A. S. Bedi, M. Bennis, and V. Aggarwal, "Energy-efficient and federated meta-learning via projected stochastic gradient ascent," *arxiv:2105.14772*, 2021.

[275] Y. Chen and M. J. Wainwright, "Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees," *arxiv:1509.03025*, 2015.

[276] S. Bahmani, P. T. Boufounos, and B. Raj, "Learning model-based sparsity via projected gradient descent," *IEEE Trans. Inf. Theory*, vol. 62, no. 4, pp. 2092 – 2099, Apr. 2016.

[277] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose, T. Ryffel, Z. N. Reza, and G. Kaissis, "Pysyft: A library for easy federated learning," 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:236690571

[278] A. Ingerman and K. Ostrowski, "Tensorflow federated," 2019. [Online]. Available: https://www.tensorflow.org/federated

[279] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1 – 207, 2019.

[280] "Tensor/io." [Online]. Available: https://doc-ai.github.io/tensorio/

[281] G. Ulm, E. Gustavsson, and M. Jirstrand, "Functional federated learning in erlang (ffl-erl)," *arxiv:1808.08143*, 2018.

[282] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4961–4973, 2021.

[283] S. Caldas, P. W. S. M. K. Duddu, T. Li, J. Konecny, V. S. H. B. McMahan, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arxiv:1812.01097*, 2018.

[284] J. Wang, C. Zhang, Y. Ding, and Y. Yang, "Awesome-META+: Meta-learning research and learning platform," *arxiv:2304.12921*, 2023.

[285] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arxiv:1811.03604*, 2018.

[286] M. J. Sheller, B. Edwards, G. A. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, and R. R. Colen, "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data," *Scientific reports*, vol. 10, no. 1, pp. 1 – 12, Jul. 2020.

[287] H. Zhang, J. Bosch, and H. H. Olsson, "End-to-end federated learning for autonomous driving vehicles," *in IJCNN*, Jul. 2021.

[288] S. Dyrmishi, R. Elshawi, and S. Sakr, "A decision support framework for automl systems: A meta-learning approach," *in ICDMW*, Nov. 2019.

[289] H. Lee, S. Li, and N. T. Vu, "Meta learning for natural language processing: A survey," *arxiv:2205.01500*, 2022.

[290] R. Kaushik, T. Anne, and J. Mouret, "Fast online adaptation in robotics through meta-learning embeddings of simulated priors," *arxiv:2003.04663*, 2020.

[291] Y. Mansour, M. Mohri, J. Ro, and A. T. Suresh, "Three approaches for personalization with applications to federated learning," *arxiv:2002.10619*, 2020.

[292] K. Oksuz, B. C. Cam, S. Kalkan, and E. Akbas, "Imbalance problems in object detection: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 10, pp. 3388 – 3415, Oct. 2021.

[293] C. Zhang, J. Bi, and P. Soda, "Feature selection and resampling in class imbalance learning: Which comes first? an empirical study in the biological domain," *Int. Conf. on Bioinformatics and Biomedicine (BIBM)*, pp. 933 – 938, Nov. 2017.

[294] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263 – 1284, Sept. 2009.

[295] L. Wang, S. Xu, X. Wang, and Q. Zhu, "Addressing class imbalance in federated learning," *In AAAI*, Feb. 2021.

[296] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *arxiv:2007.07481*, 2020.

[297] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529 – 533, 2015.

[298] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arxiv:1509.02971*, 2015.
[299] N. H. Tran, W. Bao, A. Zomaya, M. N. H. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," *in Proc. ICC*, 2019.
[300] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *arxiv:1804.05271*, 2018.
[301] M. M. Amiri, D. Gündüz, S. R. Kulkarni, and H. V. Poor, "Convergence of update aware device scheduling for federated learning at the wireless edge," *IEEE Trans. Wireless Commun.*, vol. 20, no. 6, pp. 3643 – 3658, Jun. 2021.
[302] N. Yoshida, T. Nishio, M. Morikura, K. Yamamoto, and R. Yonetani, "Hybrid-fl for wireless networks: Cooperative learning mechanism using Non-IID data," *in Proc. ICC*, 2020.
[303] T. T. Anh, N. C. Luong, D. Niyato, D. I. Kim, and L. C. Wang, "Efficient training management for mobile crowd-machine learning: A deep reinforcement learning approach," *IEEE Wireless Commun. Lett.*, vol. 8, no. 5, pp. 1345 – 1348, Oct. 2019.
[304] T. T. Anh, N. C. Luong, D. Niyato, and D. I. Kim, "Challenges, applications and design aspects of federated learning: A survey," *IEEE Access*, vol. 9, pp. 124 682 – 124 700, Sept. 2021.
[305] M. Kamp, J. Fischer, and J. Vreeken, "Federated learning from small datasets," *arxiv:2110.03469*, 2021.
[306] M. S. H. Abad, E. Ozfatura, D. Gunduz, and O. Ercetin, "Hierarchical federated learning across heterogeneous cellular networks," *in ICASSP*, 2020.
[307] H. Yuan and T. Ma, "Federated accelerated stochastic gradient descent," *arxiv:2006.08950*, 2020.
[308] S. Luo, X. Chen, Q. Wu, Z. Zhou, and S.YU, "Hfel: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6535 – 6548, Oct. 2020.
[309] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," *in ICC*, Jun. 2020.
[310] S. Kumar, P. Shah, D. H. Tur, and L. Heck, "Federated control with hierarchical multi-agent deep reinforcement learning," *arxiv:1712.08266*, 2017.
[311] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," *In ICML*, Aug. 2003.
[312] S. S. Shwartz, "Online learning and online convex optimization," *Found. Trends Mach. Learn.*, vol. 4, no. 2, pp. 107 – 194, 2012.
[313] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," *in SP*, 2017.
[314] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," *in ACM SIGSAC CCS*, Oct. 2017.
[315] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *in Proc. Int. Conf. Learn.*, 2014.
[316] F. Tramer, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," *in USENIX Security Symposium*, Aug. 2016.