INVITED PAPER

Wireless Network Intelligence at the Edge

By Jihong Park, *Member IEEE*, Sumudu Samarakoon, *Member IEEE*, Mehdi Bennis, and Mérouane Debbah, *Fellow IEEE*

ABSTRACT | Fueled by the availability of more data and computing power, recent breakthroughs in cloud-based machine learning (ML) have transformed every aspect of our lives from face recognition and medical diagnosis to natural language processing. However, classical ML exerts severe demands in terms of energy, memory, and computing resources, limiting their adoption for resource-constrained edge devices. The new breed of intelligent devices and high-stake applications (drones, augmented/virtual reality, autonomous systems, and so on) requires a novel paradigm change calling for distributed, low-latency and reliable ML at the wireless network edge (referred to as edge ML). In edge ML, training data are unevenly distributed over a large number of edge nodes, which have access to a tiny fraction of the data. Moreover, training and inference are carried out collectively over wireless links, where edge devices communicate and exchange their learned models (not their private data). In a first of its kind, this article explores the key building blocks of edge ML, different neural network architectural splits and their inherent tradeoffs, as well as theoretical and technical enablers stemming from a wide range of mathematical disciplines. Finally, several case studies pertaining to various high-stake applications are presented to demonstrate the effectiveness of edge ML in unlocking the full potential of 5G and beyond.

KEYWORDS | 6G; beyond 5G; distributed machine learning (ML); latency; on-device machine learningML; reliability; scalability; ultrareliable and low-latency communication (URLLC).

I. SIGNIFICANCE AND MOTIVATION

This research endeavor sits at the confluence of two transformational technologies, namely, the fifth generation of wireless communication systems, known as 5G [1], and machine learning (ML) or artificial intelligence. On the one hand, while the evolutionary part of 5G, enhanced mobile broadband (eMBB), focusing mainly on millimeter-wave (mmWave) transmissions has made significant progress [2], fundamentals of ultrareliable and low-latency communication (URLLC) [3], [4], one of the major tenets of the 5G revolution, are yet to be fully understood. In essence, URLLC warrants a departure from average-based system design toward a clean-slate design centered on tail, risk, and scale [5]. While risk is encountered when dealing with decision-making under uncertainty, the scale is driven by the sheer amount of devices, antennas, sensors, and actuators, all of which pose unprecedented challenges in network design, optimization, and scalability.

On the other hand, in just a few years, breakthroughs in ML and particularly deep learning have transformed every aspect of our lives from face recognition [6], [7], medical diagnosis [8], [9], and natural language processing (NLP) [10], [11]. This progress has been fueled mainly by the availability of more data and more computing power. However, the current premise in classical ML is based on a single node in a centralized and remote data center with full access to a global data set and a massive amount of storage and computing power, sifting through these data for inference. Nevertheless, the advent of a new breed of intelligent devices and high-stake applications ranging from drones to augmented reality/virtual reality (AR/VR) applications, and self-driving vehicles, makes cloud-based ML inadequate. These applications are real-time, cannot afford latency, and must operate under high reliability even when network connectivity is lost.

Indeed, an autonomous vehicle that needs to apply its brakes cannot allow even a millisecond of latency that might result from cloud processing, requiring split-second

Manuscript received December 6, 2018; revised August 22, 2019; accepted September 3, 2019. This work was supported in part by the Academy of Finland under Grant 294128, in part by the 6Genesis Flagship under Grant 318927, in part by the Kvantum Institute Strategic Project (SAFARI), in part by the Academy of Finland through the MISSION Project under Grant 319759, and in part by the Artificial Intelligence for Mobile Wireless Systems (AIMS) project at the University of Oulu. *(Corresponding author: Jihong Park.)*

J. Park, **S. Samarakoon**, and **M. Bennis** are with the Centre for Wireless Communications, University of Oulu, 90014 Oulu, Finland (e-mail: jihong.park@oulu.fi; sumudu.samarakoon@oulu.fi; mehdi.bennis@oulu.fi).

M. Debbah is with the CentraleSupélec, Université Paris-Saclay, 91190 Gif-sur-Yvette, France (e-mail: merouane.debbah@centralesupelec.fr).

Digital Object Identifier 10.1109/JPROC.2019.2941458

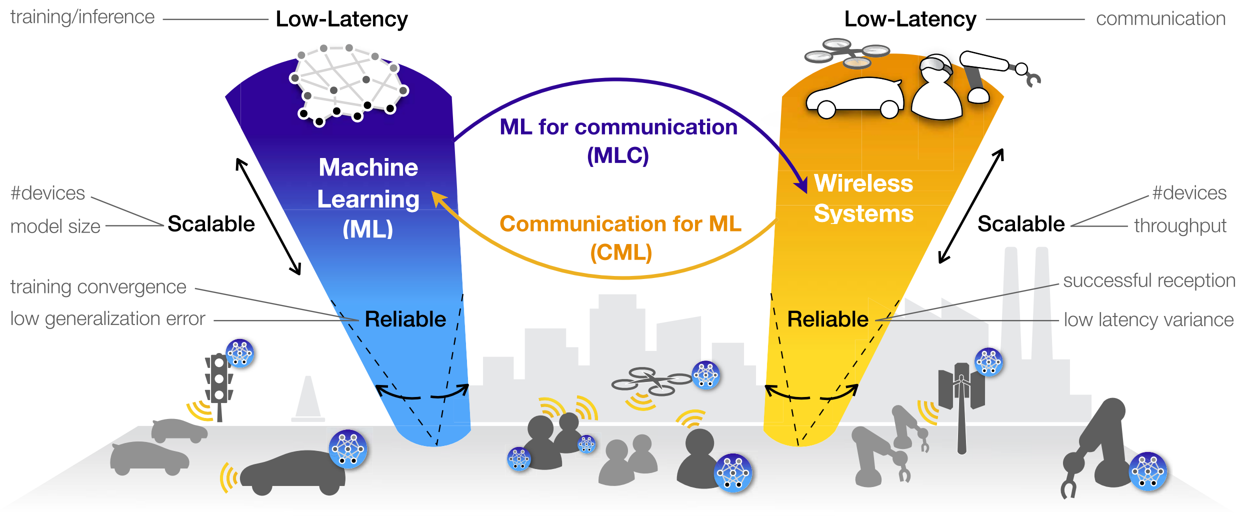


Fig. 1. ***Illustration of edge ML where both ML inference and training processes are pushed down into the network edge (bottom), highlighting two research directions: 1) MLC (from left to right) and 2) CML (from right to left).***

decisions for safe operation [12], [13]. A user enjoying visuo-haptic perceptions requires not only minimal individual perception delays but also minimal delay variance to avoid motion sickness [14], [15]. A remotely controlled drone or a robotic assembler in a smart factory should always be operational even when the network connection is temporarily unavailable [16]–[18], by sensing and reacting rapidly to the local (and possibly hazardous) environments.

These new applications have sparked a huge interest in distributed, low-latency and reliable ML calling for a major departure from cloud-based and centralized training and inference toward a novel system design coined edge ML, in which: 1) training data are unevenly distributed over a large number of edge devices, such as network base stations (BSs) and/or mobile devices, including phones, cameras, vehicles, and drones and 2) every edge device has access to a tiny fraction of the data and training and inference are carried out collectively. Moreover, edge devices communicate and exchange their locally trained models [e.g., neural networks (NNs)], instead of exchanging their private data.

There are clear advantages using edge ML.

1) Performing inference locally on connected devices reduces latency and cost of sending device-generated data to the cloud for prediction.
2) Rather than sending all data to the cloud for performing ML inference, inference is run directly on the device, and data are sent to the cloud only when additional processing is required.
3) Getting inference results with very low latency is important in making mission-critical Internet-of-Things (IoT) applications that respond quickly to local events.
4) Unlike cloud-based ML, edge ML is privacy preserving in which the training data are not logged at the cloud but are kept locally on every device, and the globally shared model is learned by aggregating locally computed updates, denoted as model state information (MSI), in a peer-to-peer manner or via a coordinating (federating) server.
5) Higher inference accuracy can be achieved by training with a wealth of user-generated data samples that may even include privacy-sensitive information, such as healthcare records, factory/network operational status, and personal location history.

Edge ML is a nascent research field whose system design is entangled with communication and on-device resource constraints (i.e., energy, memory, and computing power). In fact, the size of an NN and its energy consumption may exceed the memory size and battery level of a device, hampering decentralized inference. Moreover, the process of decentralized training involves a large number of devices that are interconnected over wireless links, hindering the training convergence due to the stale MSI exchange under poor wireless channel conditions. As such, enabling ML at the network edge introduces novel fundamental research problems in terms of jointly optimizing training, communication, and control under end-to-end (E2E) latency, reliability, privacy, as well as devices' hardware requirements. As shown in Fig. 1, these research questions can be explored through the following two research directions.

a) ML for communications: Exploiting edge ML for improving communication, on the one hand, epitomizes the research direction of ML for communication (MLC). The recent groundswell interest in the ML-aided (and mostly data-driven) wireless system design fits into this direction. At its core, MLC leverages a large amount of data samples (e.g., radio signals) to acquire an accurate knowledge of the RF environment to, for instance, optimize modulation coding schemes (MCSs). Toward this vision, at the physical layer, an E2E communication framework was studied under unknown channels using an autoencoder (AE) [19], [20], recurrent NN (RNN) [21], and generative adversarial network (GAN) [22]. To overcome mmWave's link sensitivity to blockage [23], a GAN-aided long-term channel estimation [24] and a reinforcement learning (RL)-based beam alignment technique [25] were proposed. At the network layer, an RNN-aided caching solution was proposed in [26], and an unsupervised clustering algorithm was used with real user traffic patterns. Basics of ML, its NN architectures, and communication designs were recently overviewed in [26] and [27].

Nevertheless, these works focus solely on improving the communication performance via centralized ML, ignoring the additional latency induced by the ML inference. Furthermore, they commonly presume that well-trained ML models with a large number of data samples are available, overlooking the ML model training latency. In sharp contrast to these approaches, the latency and reliability of edge ML have to be examined with respect not only to communication but also to decentralized ML training and inference processes, calling for novel analytical methods based on studying tail distributions, a novel communication and ML codesign, and uncertainty/risk assessment.

b) Communication for ML: As alluded to earlier, training ML at the network edge over wireless networks while taking into account latency and reliability opens up a novel research direction. In this respect, edge ML architectures and their operations should be optimized by accounting for communication overhead and channel dynamics while coping with several problems, such as straggling devices in the training process and generalization to unmodeled phenomena under limited local training data. In addition, all these aspects need to factor in on-device constraints, including energy, memory, and compute, not to mention privacy guarantees.

An interesting example of edge ML training architecture is federated learning (FL) [28], [29] in which mobile devices periodically exchange their NN weights and gradients during local training. FL has been shown to improve communication efficiency by MSI quantization [30], adjusting the MSI update period [28], and optimizing devices' scheduling policy [31]. These methods are still in their infancy and need to address a myriad of fundamental challenges, including the ML-communication codesign, while accounting for on-device constraints and wireless channel characteristics.

From a theoretical standpoint, the overarching goal of this article is to explore building blocks, principles, and techniques focusing on communication for ML (CML). As on-device processing becomes more powerful, and ML grows more prevalent, the confluence of these two research directions will be instrumental in spearheading the vision of truly intelligent next-generation communication systems, 6G [32].

Scope and Organization: Enabling ML at the network edge hinges on investigating several fundamental questions, some of which are briefly summarized next.

> **Q1**. How do edge devices train a high-quality centralized model in a decentralized manner, under communication/on-device resource constraints and different NN architectures?

Classical ML has been based on the precept of a single central entity having full access to the global data set over ultrafast wired connections, e.g., a local interchipset controller manipulating multiple graphics processing units (GPUs) through PCI Express intracomputer connections (supporting up to 256 Gb/s) [33], InfiniBand intercomputer links (up to 100 Gb/s) [34]; or a cloud server commanding multiple computing devices via Ethernet communication (up to 25 Gb/s) [35]. Using a deep NN, the central controller sifts through this global data for training and inference by exploiting the massive amount of storage and computing power: e.g., 11.5 Petaflops processing power supported by 256 tensor processing units (TPUs) and 4-TB high-bandwidth memory (HBM) [36], which is sufficient for operating the Inception V4 NN model consuming 44.3 GB [37].

These figures are in stark contrast with the capability of devices under edge ML. While 5G peak rates achieve 20 Gb/s [1] that is comparable only with Ethernet connections, the instantaneous rate may frequently fluctuate due to poor wireless channel conditions, hindering interdevice communication. Moreover, the computing power of mobile devices is a million times less powerful (e.g., Qualcomm Snapdragon 845's 16.6 Gflops [38]). Their memory size is also ten times smaller (Apple iPhone XS Max's 4 GB [39]) than a deep NN model size. Besides, the energy consumption (e.g., Google Pixel 3 XL's 2.15 W [40]), delimited by the battery capacity, is half million times smaller than a powerful centralized ML architecture (e.g., AlphaGo's 1 MW [41]). Since computation and communication consume battery power, the energy consumption of edge ML operations should be flexibly optimized over time, which is not feasible under the classical ML architecture, in which an NN model size is fixed.

Last but certainly not the least, the privacy guarantee of each device is crucial in edge ML, particularly when on-device data sets are associated with privacy-sensitive information. Perturbing the information exchange can ensure privacy, which may be at odds with the goal of high accuracy and reliability. Adding redundant information can provide a solution, at the cost of extra communication latency. With these communication and on-device hardware/privacy requirements in mind, we explore and propose the decentralized architectures and training algorithms that are suitable for edge ML. To this end, we first describe the key building blocks of edge ML

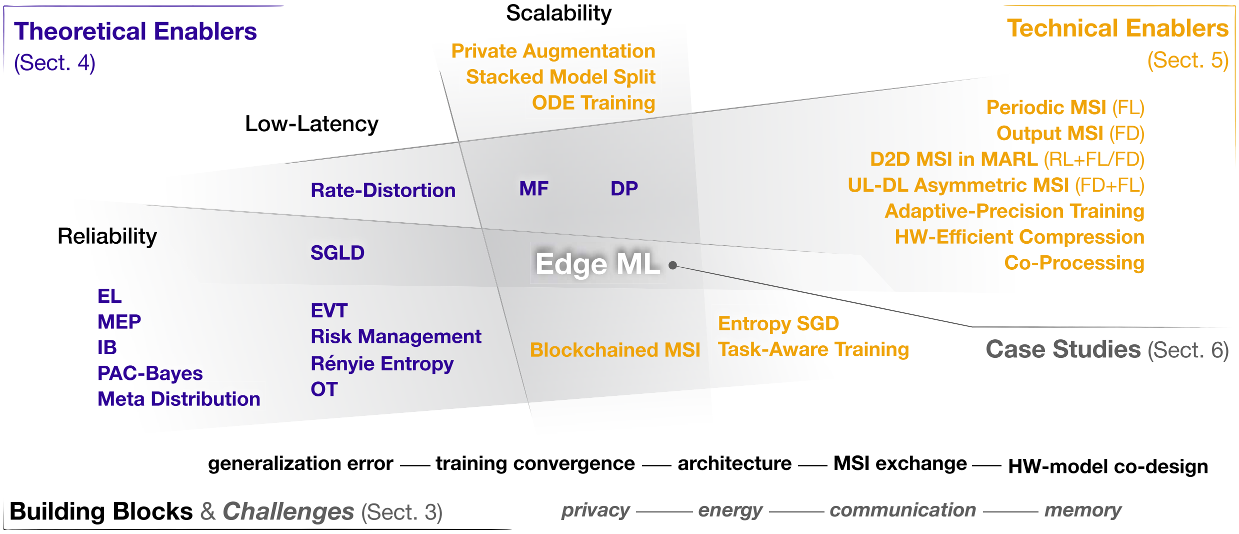


Fig. 2. ***Overview of the key building blocks and theoretical/technical enablers of edge ML.***

(see Section III) and then introduce suitable technical enablers (see Section V), as shown in Fig. 2.

> **Q2**. How to enable reliable edge ML as opposed to best-effort cloud-based ML, subject to nonconvex loss functions and unevenly dispersed and unseen data samples?

A requisite for reliable edge intelligence is high inference accuracy, i.e., low loss, under not only the training data set but also unseen data samples. Therefore, average inference accuracy is insufficient, and its credibility interval for a given training data set ought to be considered. Credibility can be measured by the loss difference under training and the entire samples, referred to as a generalization error, in which the credibility interval is an achievable target generalization error that can be decreased by utilizing more training samples and a proper NN model architecture. Calculating the generalization error is relatively easy in centralized ML where the central controller feeds independent and identically distributed (IID) training data samples into each device. By contrast, it becomes more challenging in edge ML where the training data samples may become non-IID across devices.

Next, decentralized training dynamics in edge ML become more complicated even under a simple gradient-descent algorithm. In fact, it is difficult to characterize the convergence behavior of a decentralized training process, especially under non-IID training data set, as well as the limited communication/computation resource budget fluctuating over time. The situation is aggravated when a single NN model is split and shared by multiple devices due to the limited on-device memory size.

Besides, most training algorithms intentionally insert noise, i.e., regularizers, to cope with nonconvex loss functions. However, it is challenging to optimize the regularization in edge ML, which is intertwined with wireless communication and privacy-preserving methods generating noise. To tackle these difficulties, we address Q2 by revisiting the fundamental principles of ML (see Section III), followed by the key theoretical enablers for edge ML (see Section IV).

> **Q3**. How do the theoretical and technical enablers of edge ML impact E2E latency, reliability, and scalability throughout the training and inference processes, under both CML and MLC frameworks?

From the standpoint of MLC, stringent URLLC applications can be empowered using edge ML, whereas in CML, edge ML is enhanced via wireless connectivity under on-device constraints. In this respect, edge ML design not only enhances CML but also MLC, calling for optimizing E2E latency, reliability, and scalability.

Specifically, the worst case E2E latency of a reference device is given by "training + inference + application" latency. MLC can reduce the application delays that are proportional to wireless communication latency, given as "payload/[bandwidth × spectral efficiency]." For instance, this is viable by improving spectral efficiency (SE) via enabling real-time joint source–channel coding [42], channel-agnostic E2E communications [19], and interference management [43]. The effective amount of bandwidth can also be increased using low-complexity dynamic spectrum access [44] and proactive resource management [45]. Furthermore, the payload size can be minimized by exchanging semantic information rather than raw data [46].

Achieving these benefits in MLC entails extra training and inference delays of ML operations. Here, training latency is captured by a loss or weight convergence delay [47], whereas inference latency refers to computation and memory access delays [48]. Since both the

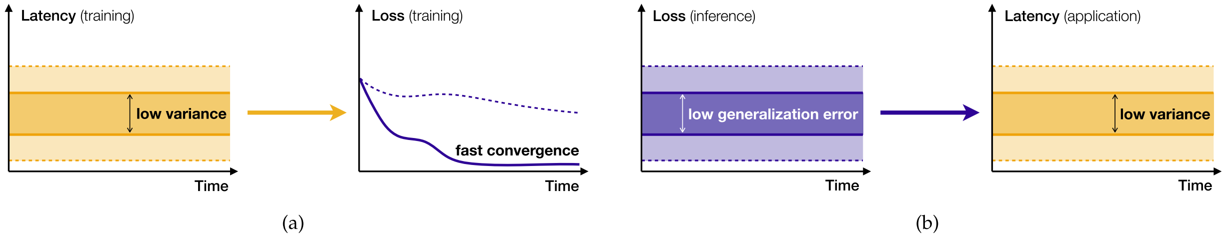


Fig. 3. *Two examples illustrating CML and MLC. (a) Training latency and variance over wireless connections impact the ML training convergence. (b) ML inference loss and variance affect the latency of a wireless system as its application.*

training and inference processes are performed at the network edge, their computation and communication need to be jointly optimized subject to limited energy, memory, computing power, and radio resource constraints. To further reduce latency, as done in [49], the training process can partly be offloaded to other proximal edge nodes with higher computational power.

E2E reliability of a reference device is the probability that the E2E latency does not exceed a target latency deadline and is determined by the reliability of both training and inference. For training reliability, the staleness of exchanged MSIs is the key bottleneck, as it disrupts the training convergence, and may lead to an unbounded training latency, emphasizing the importance of communication techniques to synchronize edge nodes. For inference reliability, high inference accuracy is mandatory for reducing the application latency and needs to be ensured under both training and unseen samples, necessitating suitable mathematical tools to quantify the generalization error.

Through E2E reliability, CML and MLC are intertwined. For CML, the higher the communication latency reliability during training, i.e., low communication latency variance, the higher the training convergence guarantee by avoiding training devices lagging behind, known as stragglers. As shown in Fig. 3(a), lower latency variance decreases the inference loss, thereby reducing the application (i.e., communication) latency and increasing E2E reliability. In the opposite direction, for MLC, a lower generalization error yields fewer fluctuations in the application latency, as visualized in Fig. 3(b), thereby achieving higher E2E reliability.

Finally, E2E scalability is specified by the number of supported devices, NN model size, and communication throughput to ensure a target E2E reliability. In this respect, the major obstacles are on-device constraints, whereby the number of federating devices is limited by preserving their privacy. The range of federation is also determined by memory and NN model sizes, as FL requires an identical model size for all federating devices. Moreover, the federation range should take into account tasks' correlations and channel conditions, so as not to exchange redundant MSIs and to avoid straggling devices under limited wireless capacity, respectively.

With these challenges and E2E performance definitions, to address Q3, we revisit the state-of-the-art literature in URLLC and ML (see Section II) and provide several case studies that showcase the essence of both MLC and CML frameworks (see Section VI), followed by the conclusion (see Section VII).

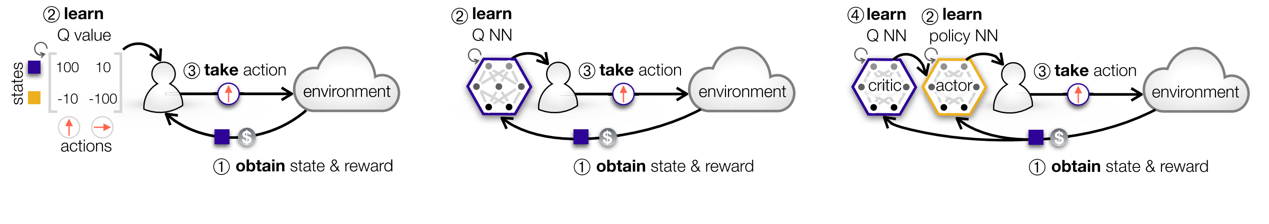
II. STATE OF THE ART

A. From Vanilla 5G Toward URLLC Compound

Since its inception, 5G requirements have been targeting three generic and distinct services: eMBB, mMTC, and URLLC [50]–[52]. The prime concern of eMBB is to maximize SE by providing higher capacity for both indoor and outdoor highly dense areas, enhancing seamless connectivity for everyone and everywhere, and supporting high mobility, including cars, trains, and planes. Beyond voice and data services, eMBB focuses on immersive AR/VR applications with high-definition 360° video streaming for entertainment and navigation. Enabling communication among a large number of devices is the focus of mMTC. The success therein relies on coverage, low power consumption, longer life span of devices, and cost efficiency. The goal of URLLC is to ensure reliability and latency guarantees. Therein, mission-critical applications, including autonomous driving, remote surgery, and factory automation, require outage probabilities of the range from 10^{-5} to 10^{-9}.

Having said that, recent field tests have shown that packet sizes of less than 200 bytes were supported for a single device moving within 1 km [53]. This demonstrates the fundamental challenge in achieving (even moderate) URLLC requirements, e.g., 99.999% decoding success rate with 1-ms latency [54]. On the other hand, new emerging use cases advocate a mixture of URLLC with eMBB and/or mMTC via slicing, in which a pool of shared resources (spectrum, computing power, and memory) is used [55]–[61].

The success behind fine-grained and dynamic network slicing hinges on the ability to identify and distinguish different services within the network. Unfortunately, such traditional slicing methods are unfit for the upcoming high-stake applications that may not only demand more stringent requirements but also give rise to a compound of URLLC with eMBB and/or mMTC, where a single device simultaneously requests several services out of eMBB, mMTC, and URLLC. For instance, multiplayer mobile



(a) (b) (c)

Fig. 4. ***Examples of RL. (a) Classical Q-learning without any NN. (b) Deep Q-learning with an NN. (c) Actor-critic RL with actor and critic NNs.***

AR/VR gaming applications demand URLLC for achieving low motion-to-photon (MTP) latency and eMBB for rendering 360° video frames. Likewise, autonomous vehicles rely on sharing high-resolution real-time maps with low latency, which is an exemplary use case of ultrahigh-speed low-latency communications (uHSLLC) [62]. In a similar vein, mission-critical factory automation applications may require sensing, reasoning, and perception-based modalities (e.g., haptic), going beyond classification tasks [63], necessitating both eMBB and URLLC.

Beyond communication, automated or remotely controlled vehicles, drones, and factories demand ultrareliable and low-latency control. A swarm of remotely controlled drones affected by a sudden gust of wind may need to report and receive control commands within a split of a second. Likewise, a remote surgery robot operating a critical patient requires a high-definition video signal upload while receiving real-time commands of high precision that are two such examples. The reliability of the control feedback loop of the aforementioned control systems directly affects the system performance. Both lost or outdated information and commands can yield undesirable and chaotic system behaviors in which the communication links between controllers and devices play a pivotal role. In this regard, the research in URLLC and control has recently emerged focusing on coordination [64]–[66], robustness [67]–[69], and sensing [70], [71].

The existing orthogonal and nonorthogonal slicing approaches are ill-suited for supporting these compounded URLLC services. Compared with a noncompounded scenario, orthogonal slicing under compounded URLLC consumes the resource amount multiplied by the number of compounded links per device. Due to the use of multiple links by a single device, nonorthogonal slicing induces severe multiservice self-interference [14], negating the effectiveness of slicing. These limitations call for the aid from a new dimension, namely, edge ML as elaborated in Section II-B.

B. From Centralized ML Toward Edge ML

1) Types of ML: The training process of ML is categorized into supervised, unsupervised, and RL as follows.

a) Supervised learning: By feeding an input data sample, the goal of supervised learning is to predict a target quantity, e.g., regression, or classification of the category within the predefined labels. This ability can be obtained by optimizing the NN parameters by feeding training data samples, referred to as a training process. In supervised learning, the input training samples are paired with the ground-truth output training samples. These output samples "supervise" the NN to infer the correct outputs for the actual input samples after the training process is completed.

b) Unsupervised learning: The training process of unsupervised learning is performed using only the input training samples. In contrast to supervised learning, unsupervised learning has no target to predict, yet it aims at inferring a model that may have generated the training samples. Clustering of ungrouped data samples and generating new data samples by learning the true data distribution, i.e., a generative model, belong to this category.

c) Reinforcement learning: The goal of RL is to make an agent in an environment taking an optimal action at a given current state, where the interaction between the agent's action and the state through the environment is modeled as a Markov decision process (MDP). When each action is associated with a return, the agent takes an action that maximizes its predicted cumulative return, e.g., Q-learning that maximizes the Q value for each state, as shown in Fig. 4(a). In Q-learning, the larger the state dimension, the more computation. This problem is resolved by deep Q-learning as shown in Fig. 4(b), where an NN approximates the Q function and produces the Q values by feeding a state. These value-based RL can take actions only through Q values that are not necessarily required. Instead, one can directly learn a policy that maps each state into the optimal action, which is known as policy-based RL whose variance may become too large [72]. Actor-critic RL is a viable solution to both problems, comprising an NN that trains a policy (actor NN) and another NN that evaluates the corresponding Q value (critic NN), as visualized in Fig. 4(c).

For the sake of convenience, unless otherwise specified, we hereafter describe ML from the perspective of supervised learning, most of which can also be applied to unsupervised and RL algorithms with minor modifications.

2) Types of NN Architectures: Fig. 5 shows an NN consisting of multiple layers. The input layer accepts input

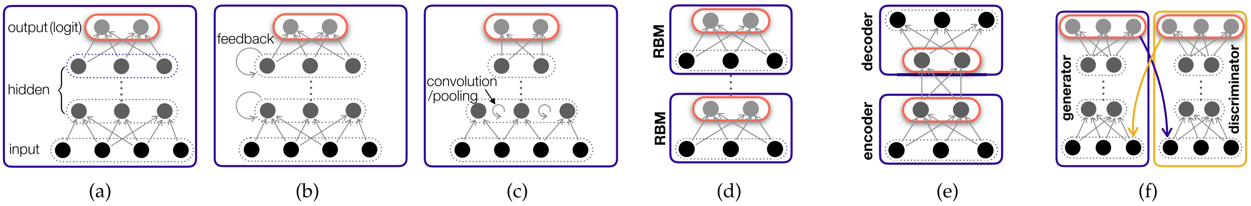


Fig. 5. *Types of NN architectures. (a) MLP. (b) RNN. (c) CNN. (d) DBN comprising a stack of pretrained RBMs. (e) AE. (f) GAN.*

data samples, and the output layer produces the training/inference outcomes. These two layers are connected through at least a single hidden layer. The total number of hidden layers is called the depth of the NN. An NN with a large depth is referred to as a deep NN, otherwise, a shallow NN.

Each of the layers comprises perceptrons connected to each other, and such connections are associated with weights. At a single perceptron, all the inputs are thus weighted and aggregated, followed by the output value after passing through a nonlinear activation function, e.g., sigmoid function, softmax function, or rectified linear unit (ReLU). The output is fed to the next layer's perceptrons until reaching the output layer.

a) Multilayer perceptron: If the output of each layer is fed forward to the next layer, and then, the NN is called a feedforward NN (FNN). The default (i.e., vanilla) baseline of the FNN is a multilayer perceptron (MLP). As shown in Fig. 5(a), each perceptron's output is passed directly to the next layer's perceptrons, without any recursion and/or computation other than the activation function. Even with this simple structure, an MLP is capable of distinguishing data that are not linearly separable, as long as the number of perceptrons (i.e., NN model size) is sufficiently large. Its theoretical backgrounds are elaborated in Section III-B1. Training an MLP is performed using a gradient descent optimization algorithm, called the backpropagation method [73].

b) Recurrent NN: If the output of a layer encounters recursions within the same layer, the corresponding NN is referred to as an RNN. As shown in Fig. 5(b), the vanilla RNN is a form of MLP with feedback loops at hidden layers. For training an RNN, the feedback loops can be unrolled in a sequential way, such that the hidden layer prior to a feedback loop is chained to the hidden layer posterior to the loop. This chained structure easily allows a sequence of inputs, which is plausible, for instance, in NLP. Vanilla RNN struggles with the vanishing gradients problem as the feedback loop iterates and thus cannot capture a feature with long-term correlations. To cope with this, the vanilla RNN's hidden layer can be replaced with a long short-term memory (LSTM) unit [74]. The LSTM unit introduces a memory cell that can store the current hidden layer's values in the memory, which is controlled by several gates that determine whether to store or forget. Similar operations can also be implemented using gates, yielding a gated recurrent unit (GRU) [75].

c) Convolutional NN: Processing image data samples through an MLP may induce a large number of perceptrons, as each image pixel needs to be associated with a single perceptron in the input layer. As a variant of MLPs, a convolutional neural network (CNN) resolves this problem by inserting two preprocessing layers, i.e., convolutional and pooling layers, in-between the hidden layers, as shown in Fig. 5(c). Inspired by human visual stimuli, at a convolutional layer, the input information is processed using a convolution operation, thereby extracting the features while compressing the information. Next, at the pooling layer, the previous layer's outputs are combined into a single perceptron in the next layer, by selecting their maximum value or taking their average value [76]. Finally, the compressed feature information is fed back to a conventional MLP structure to obtain the final output, of which the corresponding stages are called fully connected layers.

d) Deep belief network: A notorious problem of training a deep NN is the vanishing gradients as the depth increases. To resolve this, a deep belief network (DBN) exploits a divide-and-conquer method: first pretrain each small part of the network, i.e., restricted Boltzmann machine (RBM), and then combine all pretrained parts, followed by fine-tuning the entire network. As a result, a DBN comprises a stack of pretrained RBMs, as shown in Fig. 5(d). Each RBM has a single hidden layer and a visible layer that accepts inputs or produces outputs. The connections between these two layers are bidirectional, in which an RBM is not an FNN. Nonetheless, after the pretraining process, the pretrained RBMs in a DBN are stacked in a way that each RBM's hidden layer connects no longer to its visible layer but to the next RBM's visible layer. Thus, a DBN is an FNN. DBN can easily be extended to a CNN by partitioning each hidden layer into several groups and applying a convolutional operation to each group [77].

e) Autoencoder: As shown in Fig. 5(e), an AE is a stack of two FNNs, copying the input to its output in an unsupervised learning way. The first NN learns representative features of the input data, known as encoding. The second NN receives the feature as the input and reproduces the approximation of the original input as the final output, referred to as decoding. The key focus of an AE is to learn useful features of the input data en route for copying the input data. In this respect, reproducing the output exactly the same as the original data may become too accurate to capture the latent features. A viable solution for mitigating this is to constrain the feature space to have a smaller dimension than the original input space, yielding an undercomplete AE whose encoding NN compresses the original input data into a short code [78]. Decoding the NN then uncompresses the code so as to approximately reproduce the original data. These encoding and decoding NNs resemble a transmitter and its receiver in communication systems. Due to such an analogy, AEs have recently been used for data-driven E2E communication designs [19], [20], after inserting a set of extra layers that emulate the transmission power normalization and the channel propagation between the encoding and decoding NNs.

f) Generative adversarial network: As a generative model in the class of unsupervised learning, the goal of GAN is to generate new data samples given by the estimated distribution of the input data samples. This is achieved by training two NNs, referred to as a generator and a discriminator as shown in Fig. 5(f), as if they play a zero-sum game. Here, the generator produces fake samples to fool the discriminator, while the discriminator tries to identify the fake samples. As the game reaches a Nash equilibrium, i.e., training completion, the generator becomes capable of producing fake-yet-realistic samples that are indistinguishable from the real samples. Mathematically, playing this game is identical to minimizing the Jensen–Shannon (JS) divergence [79] between the real and generated sample distributions. When these two distributions are disjoint, JS divergence goes to infinity, yielding the gradient vanishing problem. A naive solution is to insert noise so as to make the distributions overlapped, which may degrade the output quality. A better solution is to replace JS divergence with a proper loss function that is capable of measuring the distance between the disjoint distributions with a finite value. Wasserstein distance satisfies this characteristics, thus prompting the recent success of Wasserstein GAN (WGAN) [80]. More details on the Wasserstein distance are deferred to Section IV-A.

3) Limitations of Centralized ML: The classical concept of ML focuses mainly on offline and centralized ML [21], [26]. In this case, the entire data set is given *a priori* and is used for the training process. To obtain accurate and reliable inference, a central controller divides the training data set into minibatches and allocates them to multiple processing devices, e.g., via a message passing interface (MPI) [81], running local training operations. The central controller iteratively collects and aggregates their local training results until the training loss converges. The said training process is separated from inference, and hence, the training cost and latency are commonly neglected.

Unfortunately, such a one-time training process is vulnerable to initially unmodeled phenomena, as it is practically impossible to enumerate all preconditions and ensuing consequences, referred to as qualification and ramification problems, respectively [82]. In fact, the trained model using centralized ML is biased toward the initially fixed training data set, which fails to capture user-generated and the time-varying nature of data, yielding less reliable inference results. Besides, a user-generated data set can be privacy-sensitive, and the owners may not allow the central controller to directly access the data. Online and edge ML is able to address these problems. Indeed, online decentralized training can preserve privacy by exchanging not the data set but the model parameters with (or without) a simple parameter aggregator [29], thereby reflecting a huge volume of user-generated data samples in real time. In so doing, the trained models are immediately obtained at the local mobile devices, enabling low-latency inference. In spite of the rich literature in ML, edge ML with a large number of mobile devices over wireless remains a nascent field of research, motivating us to explore its key building blocks and enablers as elaborated in Section III.

III. BUILDING BLOCKS AND CHALLENGES

A. Neural Network Training Principles

The unprecedented success of ML has yet to be entirely demystified. Till today, it is not completely clear how to train an NN so as to achieve high accuracy even for unseen data samples. Nonetheless, recent studies on the asymptotic behaviors of a deep NN training process and on the loss landscape shed some light on providing guideline principles, presented in Sections III-A1 and III-A2.

1) Asymptotic Training Principles:

a) Universal approximation theorem: The theorem states that an ideally trained MLP with infinitely many perceptrons can approximate any kind of nonlinear function. This holds irrespective of the number of hidden layers. Thus, both very-wide shallow NNs and very-deep NN can become ideal classifiers. The proof is provided for shallow NNs [83], [84] and recently for deep NNs [85]. This explains that the key benefit of deep NNs compared to shallow NNs is attributed not only to the inference accuracy but also to its credibility, as detailed next.

b) Energy landscape (EL): The success of deep NNs has recently been explained through the lens of energy landscape (EL) from statistical physics. In this approach, a fully connected FNN with L layers is first transformed into a spherical spin-glass model with L spins, and the FNN's nonconvex loss function is thereby approximated as the spin-glass model's nonconvex Hamiltonian [86], i.e., energy of the NN model. Exploiting this connection, a recent discovery [87] verifies that the number of Hamiltonian's critical points is a positively skewed unimodal curve over L. For a sufficiently large L in a deep NN, the number of local minima and saddle points thus monotonically decreases with L. At the same time, it also verifies that Hamiltonian's local minima become more clustered in EL as L increases. For these reasons, in a deep NN, any local minimum of a nonconvex loss function well approximates the global minimum. This enables to train a deep NN using a simple gradient-descent approach. In other words, a deep NN training can

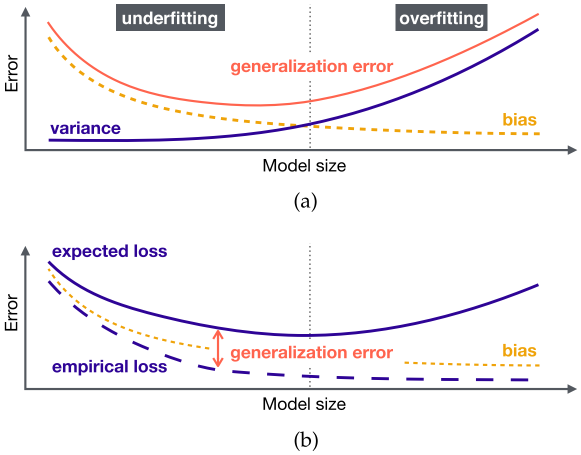


Fig. 6. ***Illustration of underfitting and overfitting via bias-variance tradeoff and approximation-generalization tradeoff. (a) Tradeoff between bias and variance whose addition yields a generalization error. (b) Tradeoff between empirical average and expected average whose difference yields a generalization error.***

always achieve the highest inference accuracy, so long as a sufficiently large amount of data samples are fed for training, emphasizing the importance of sufficient data sample acquisition.

2) Practical Training Principles: The aforementioned asymptotic characteristics of a deep NN make the nonconvex training problem being in favor of simple convex optimization methods. This partly justifies the use of de facto stochastic gradient descent (SGD) algorithms in modern ML applications. Nevertheless, the number of layers is finite in reality, and the resultant mismatch between the nonconvex problem and its convex-based training algorithm has to be considered in practice, as elaborated next.

a) Underfitting versus overfitting: The objective of training an NN is twofold. The first goal is to minimize the training loss, and the other goal is to minimize the loss difference between the values under training data samples and unseen samples, referred to as mitigating underfitting and overfitting, respectively. How to achieve both goals is exemplified in the following bias–variance tradeoff. Consider $f(x)$ is the true function to be estimated using an approximate function $\hat{f}(x)$ for an unseen data sample x. When the distribution of x has zero mean and variance σ^2, the mean squared error of $f(x)$ and $\hat{f}(x)$ can be decomposed into bias and variance terms as follows:

$$\mathsf{E}[(f(x)-\hat{f}(x))^2] = \text{Bias}^2 + \text{Var} + \sigma^2 \tag{1}$$

where $\text{Bias} = \mathsf{E}[\hat{f}(x) - f(x)]$ and $\text{Var} = \mathsf{E}[\hat{f}(x)]^2 - (\mathsf{E}[\hat{f}(x)])^2$. For a given training data set, as the model size grows, Bias decreases, whereas Var keeps increasing, as visualized in Fig. 6(a). The sum of Bias and Var is denoted as generalization error, and both underfitting and overfitting can be mitigated at the minimum generalization error.

Such a behavior can also be characterized through the lens of the approximation–generalization tradeoff as follows. Let $\hat{L}(w)$ denote the empirical average loss under the training data set. This empirical loss decreases as the NN better approximates the features of the training data set. On the other hand, $L(w)$ represents the expected loss under the entire data set with a set w of model parameters, which depends not only on the training samples but also on unseen samples, i.e., generalization. As shown in Fig. 6(b), $\hat{L}(w)$ decreases with the model size, while $L(w)$ is convex-shaped. In this case, both underfitting and overfitting can be avoided at the minimum $L(w)$, and the gap between $L(w)$ and $\hat{L}(w)$ implies the generalization error. It is noted that calculating $L(w)$ requires unseen data samples, which is unavailable in practice. Instead, the probability that the generalization error is less than a certain value can be evaluated using statistical learning frameworks, as detailed in Section IV-A.

b) Information bottleneck principle: The bias–variance tradeoff is well observed in the NN training process under information bottleneck (IB) [88], [89]. In this approach, denoting X and Y as a random input and its desired output, respectively, model training is interpreted as adjusting the model configuration $\hat{X}$ so that its predicted output $\hat{Y}$ can be close to Y, as illustrated in Fig. 7(a). A convincing model training strategy is to minimize the redundant information $I(\hat{X};X)$ for predicting Y while maximizing the prediction relevant information $I(\hat{X};Y)$. This is recast by solving the following Lagrangian optimization problem:

$$\min_{p(\hat{x}|x),\, p(y|\hat{x}),\, p(\hat{x})} \quad I(X;\hat{X}) - \beta I(\hat{X};Y) \tag{2}$$

such that $Y \rightarrow X \rightarrow \hat{X} \rightarrow \hat{Y}$, where $\beta > 0$ is a Lagrangian multiplier. In the objective function (2), the first term $I(X;\hat{X})$ decreases with the level of model generalization, while the second term $I(\hat{X};Y)$ increases with the inference performance, thereby capturing the bias–variance tradeoff. Furthermore, the problem can be interpreted as passing the information in X about Y through a bottleneck $\hat{X}$ [90]. To illustrate, following the data processing inequality [91] in the information theory, the second term $I(\hat{X};Y)$ is upper bounded by $I(X;\hat{X})$ that coincides with the first term $I(X;\hat{X})$. Such a conflict in the optimization process incurs an IB, thereby leading to two model training phases. As shown in Fig. 7(b) under the tanh activation, during the first phase, both $I(X;\hat{X})$ and $I(\hat{X};Y)$ increase, whereas during the second phase, $I(\hat{X};Y)$ increases and $I(X;\hat{X})$ decreases. This unfolds the microscopic model training behaviors: model training tends to first increase the inference performance while relying highly on its own input data samples and then tries to generalize the model by reducing the impact of its own data samples. Nonetheless, the two-phase training dynamics are not always manifested since the original IB formulation is sensitive to the types of hidden layer activations [see Fig. 7(b)] [92] and the training objective function [93], which can be partly rectified by modifying the objective function and/or inserting noise [94].

c) Flat minima: Although deep NNs asymptotically guarantee easily trainable characteristics [87], our practi-

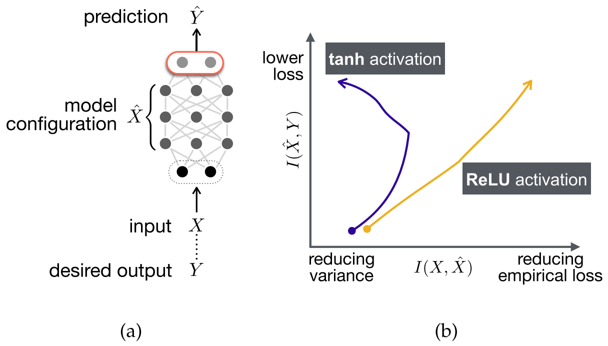


Fig. 7. ***Illustration of (a) model training structure and (b) trace of $\{I(X, \hat{X}), I(\hat{X}, Y)\}$ during model training under the IB principle.***

cal concern is the deep NN with finite depth L that cannot make all local minima become negligibly separated. In the EL shown in Fig. 8, some of the local minima are clustered in a wide valley, i.e., flat minima [95], whereas a sharp minimum [96] isolated from other minima may have lower energy than the flat minima. At this point, the well-known problem of generalization is in order. Each sharp minimum represents the overfitting case that achieves the minimum energy only at the given data and/or model configuration. In order to train a generalized model, it is thus preferable to find a flat minimum at each training iteration [95], [97]–[99].

d) Minimum energy path: Finding flat minima by using EL has recently been tackled by a counterexample [96]. Namely, while producing the same final output, it is possible to rescale the model weights, changing the EL. In this case, one can imagine a sufficiently generalized NN whose trained weights lie within flat minima. Rescaling this original NN can produce a set of modified weights lying not within flat minima, and both observations contradict with each other. This calls for a careful observation on the EL. On this account, a filter normalization technique [99] has been proposed, which heuristically guarantees a tight relationship between the resulting landscape and the final output. A recent study [100] detours this problem by using a minimum energy path (MEP) method in statistical physics. In this approach, from a given weight to another arbitrary weight, the goal is to find the path that follows the saddle points in the EL, implying the path minimizing the maximum energy. Such a path does not allow the weight rescaling operation, thereby negating the aforementioned counterexample. Furthermore, the result shows that the MEPs of widely used deep NNs are almost the same as the local minima, uncovering their generalization excellence through the lens of the EL.

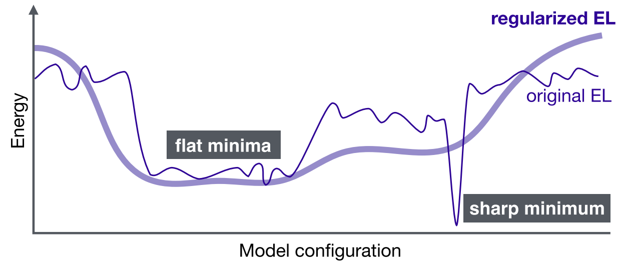


Fig. 8. ***Flat versus sharp minima and the impact of regularization in EL.***

e) Regularization and ensembling: In practice, the goal of high inference accuracy with the low generalization error is commonly achieved by heuristically inserting noise during the training phase. This smoothens out the EL as observed in Fig. 8, thereby allowing a gradient-descent method to achieve the desired objective. To this end, one can add a regularizer term to the nonconvex loss function and/or insert noise into the weight update process. The latter can be implemented indirectly by randomly sampling the training data samples in SGD or by aggregating and averaging the weight parameters computed by multiple devices, referred to as ensembling or bootstrap aggregating (bagging) [78].

B. Distributed Training

The NN training process occupies the majority of the E2E latency budget and impacts the inference reliability while delimiting the overall scalability. Toward achieving these E2E targets, parallelizing the training process is crucial. Ideally, the distributed training process can exploit a larger amount of aggregate computing power with more training data samples. In Sections III-B1 and III-B2, we discuss how to parallelize the architecture and carry out distributed training.

1) Architectural Split: An NN training process can be split by parallelizing the training data samples to multiple devices that have an identical NN structure, referred to as data split. Alternatively, when an NN model size is too large, a single NN structure can be split into multiple segments that are distributed over multiple devices, i.e., model split.

a) Data split: In centralized ML, a central controller, hereafter denoted as a master, owns the entire set of training data samples and feeds a small portion of the entire data set, i.e., a minibatch, into each subordinated training device at each round, i.e., an epoch. Afterward, the master collects and aggregates the trained model parameters, as shown in Fig. 9(a). This master-device (m-d) split corresponds to the case with a cloud server fully controlling its associated devices. The parallel computation inside a single device also fits with this split, where a controller handles multiple training processors, e.g., GPUs, inside a single device. Such a centralized training operation is ill-suited for online decentralized training in edge ML, where training data samples are generated and owned by local devices. Even in offline learning, some private data samples, e.g., patient records, may not be accessible to the master, obstructing the training process.

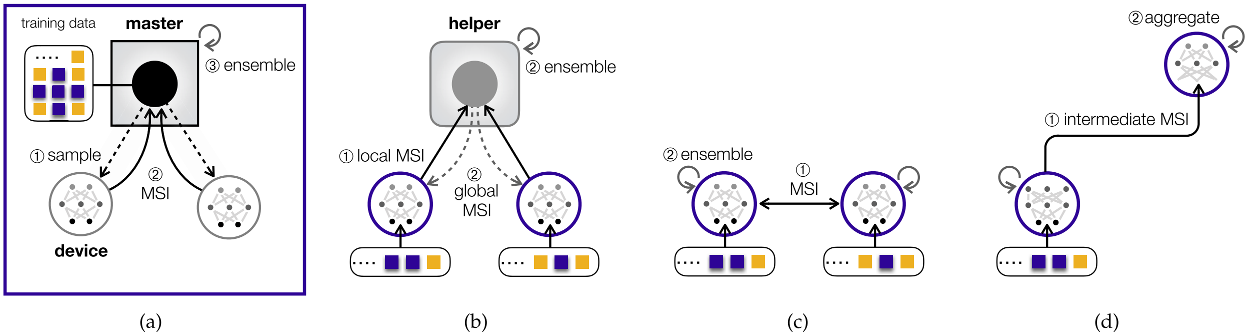


Fig. 9. ***Centralized ML with (a) master–device (m-d) split, compared to edge ML with (b) helper–device (h-d) split, (c) device–device (d-d) split, and (d) model split. The master node in centralized ML has full access to the data set and controls all devices within the blue square, whereas the helper node in edge ML can only exchange MSIs with each device having access to its local data samples.***

In edge ML, one can instead consider a helper–device (h-d) split, where each device first trains the local model using its own data samples and then exchanges the trained local model parameters with a helper that aggregates the parameters uploaded from multiple devices. Downloading the aggregated model parameter to each device completes a single epoch, as shown in Fig. 9(b). A well-known example of this h-d split is the structure with a parameter server that assists the model parameter exchanges [101]–[104]. A natural extension of this is a device–device (d-d) without any coordinator, as visualized in Fig. 9(c). The communication among devices can be enabled by a push–pull gossip algorithm [104] or by following a predefined communication network topology [105], [106]. In the context of edge ML, we henceforth focus on the h-d split and the d-d split, which is to be elaborated in Section V-B.

b) Model split: If an NN model size is larger than the device memory, the model has to be split into segments distributed over multiple devices, as shown in Fig. 9(d). In this case, devices need to exchange intermediate model parameters during the forward and backward training operations, requiring sophisticated pipelining of the processes. In order to minimize the dependence among the split segments, one can first unroll the original NN model and construct its dataflow graph [107], [108]. Then, the processing efficiency is maximized by, for example, grouping and merging common operations in the forward and backward processes, respectively. In this article, we mainly focus on the data split architecture, unless otherwise specified.

2) MSI Exchange: For a given data split, each device first trains its local model and then exchanges its current MSI. In what follows, we exemplify which MSI is exchanged and how to update the local model based on the exchanged MSI in centralized ML with the m-d split, followed by their limitation from the edge ML's standpoint.

a) Centralized parallel SGD: This baseline method, also known as full-synchronous SGD or minibatch SGD [101]–[104], exchanges the gradients of the training loss function. To elaborate, every device uploads its local MSI to the master and subsequently downloads the master's global MSI for the next local weight update. Local MSI is each device's local gradients, and global MSI is the mean gradients averaged over all devices. At the kth epoch, the ith device is fed with a randomly selected group of data samples, i.e., a minibatch, by the master. Its local weight $w_k^{(i)}$ is updated as

$$w_{k+1}^{(i)} = w_k^{(i)} - \eta \bar{g}_k \tag{3}$$

where $\bar{g}_k = 1/M \sum_{j=1}^{M} g(w_k^{(j)})$ is the gradient averaged over M devices and $g(w_k^{(j)})$ is the jth device's gradient. The parameter η is the learning rate that should decrease with k in order to compensate the weight update variance induced by the random minibatch sampling process.

b) Elastic SGD: Instead of exchanging gradients, the weights of an NN model during training can be exchanged. A typical example is elastic SGD (ESGD), where the local MSI is each device's local model weights, and the global MSI is the mean weights averaged over all devices [109]. In addition, ESGD focuses on guaranteeing the mean weight convergence, i.e., global MSI reliability, which affects all devices' local MSIs. To this end, the master in ESGD updates the global MSI not only using the current average weight but also based on the previous average weight. The local weight update of the ith device is given as

$$w_{k+1}^{(i)} = (1-\alpha)w_k^{(i)} - \eta g\big(w_k^{(i)}\big) + \alpha \hat{w}_k \tag{4}$$
$$\hat{w}_k = (1-\beta)\hat{w}_{k-1} + \beta \bar{w}_k \tag{5}$$

where $\bar{w}_k$ is the average weight at the kth epoch, and $\alpha,\ \beta < 1$ are constants.

In both centralized parallel SGD (CSGD) and ESGD, the MSI payload size is proportional to the model size and is exchanged every epoch for all devices. This becomes challenging in edge ML where the wireless communication cost is not negligible. In addition, their global MSI-based weight update rule fundamentally relies on an ensembling technique that is suitable for IID training data sets of devices. Therefore, it becomes less effective in edge ML where the user-generated training data samples may be non-IID. These issues are tackled in Sections V-B and VI-D.

C. Hardware-Model Codesign

Modern computational devices are equipped with multiple types of memory. On-chip cache, such as static random access memory (SRAM), has the smallest size, yet it is the fastest, e.g., 24 MB with tens of nanoseconds access latency [110], [111]. Dynamic random access memory (DRAM) is larger but slower than SRAM, e.g., 16 GB with hundreds of nanoseconds access latency [111]. Nonvolatile memory, such as solid-state drive (SSD), is the largest but the slowest, e.g., 30 TB with several microseconds access latency [112].

Compressing the size of an NN model makes the model fit into smaller and faster memory, empowering low-latency inference and training. In addition, model compression improves energy efficiency, since the number of memory accesses is the major source of NN's energy consumption, which is proportional to the model size [48], [113]. Finally, in distributed training, model compression minimizes the MSI payload size, thereby reducing communication latency. Model compression needs thus to consider communication and computation aspects, which is commonly achieved by the following approaches.

1) Quantization: Training an NN accompanies a large number of simple arithmetic operations, of which the intermediate calculations need to be stored in the memory. Therefore, it is effective to properly reducing the arithmetic precision of model parameters during the training process. In this respect, mixed-precision training [114] is a viable solution, where the precision is downconverted from floating point (FP) 32 to FP 16 during the forward and backward propagation processes and then upconverted back to FP 32 when updating the master copy of model weights. In so doing, memory consumption during the training is halved while not compromising accuracy compared to the training with single-precision FP 32.

2) Pruning: A deep NN commonly has a large amount of redundant weights and connections. Partially pruning them helps compress the model while maintaining the original inference accuracy. In this direction, the simplest pruning is dropping a set of perceptrons by setting their final activations to 0, known as DropOut [115], which removes all the subordinated connections of the pruned perceptrons. Alternatively, one can only prune the connections by setting the weights to 0, referred to as DropConnect [116]. The pruning can be performed uniformly randomly until the target accuracy is maintained or in a more sophisticated way, e.g., based on the Fisher information of the weights [117].

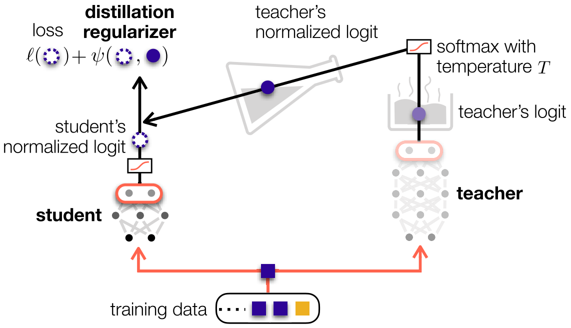


Fig. 10. ***KD that transfers a teacher model's knowledge to its student model.***

3) Knowledge Distillation: Compared to pruning that gradually compresses the model size, knowledge distillation (KD) first constructs an empty NN, referred to as a student NN, with a target compressed model size and fills in its weight parameters [118]. KD focuses on training the student NN, while both student and another pretrained teacher NN observe an identical data sample, as shown in Fig. 10. Each NN's prediction of the sample quantifies its current knowledge, and KD transfers the knowledge from the teacher to the student. For the knowledge measurement, instead of the final prediction output, e.g., "dog" in a dog-or-cat problem, KD utilizes the model output prior to the final activation, called logit values containing the distributional information of the prediction, e.g., $\{\text{dog}, \text{cat}\} = \{80, 20\}$.

The key idea of KD is adding a distillation regularizer to the student's loss function. This regularizer inserts a random noise into the student's locally calculated weight, more distorting the weight calculation when the teacher's prediction is more dissimilar to the student's prediction that is likely to be wrong. In this respect, the distillation regularizer is proportional to the knowledge gap between the teacher and the student, and the gap is commonly measured using cross entropy. Since cross entropy only accepts 0–1 ranged values, the original logit values are fed to the distillation regularizer, after being normalized via a softmax function with temperature $T \geq 1$, i.e., $\exp(z_i/T)/\sum_j \exp(z_j/T)$ for the ith label logit z_i. The increase in T smoothens the logit values, helping more information being transferred from the teacher particularly when the teacher's logit distribution is peaky. Afterward, the student's final activation is set as another softmax function without temperature, i.e., $T = 1$, which resembles a distilling process, as visualized in Fig. 10. By default, pruning and KD are performed separately after training completes. The resultant extra latency and memory usage may not well correspond to edge ML, calling for different techniques to be discussed in Section V.

D. Grand Challenges

From a theoretical standpoint, unraveling the black-box operations of ML has focused primarily on centralized ML architectures. Investigating edge ML architectures is, therefore, a daunting task, notably with the model split, where the analysis becomes extremely cumbersome due to the data flow dependence among the split model segments. Moreover, as each device feeds the data samples that can be non-IID, most of the analytical frameworks built upon IID data samples fall short in measuring the reliability of edge ML systems.

From a technical perspective, the aforementioned training and model compression approaches cannot be straightforwardly applicable to edge ML but encounter key challenges due to devices' characteristics and their inter-device communication links, as detailed next.

1) On-Device Constraints:

a) Energy limitation: In a large-scale ML system, each device is likely to be mobile, equipped with a capacity-limited battery. Thus, the finite energy needs to be efficiently utilized for the computation process during training and inference operations, as well as the communication process. In this respect, one may prefer to slightly decrease the inference accuracy in order to save the training energy consumption that is proportional to the model size [48]. Unfortunately, traditional ML architectures cannot dynamically change the model size in real time and are thus unable to flexibly optimize the energy consumption. Next, with the helper–device split, the device may want to offload its computation to the helper with higher computation power without energy limitation. The bottleneck is the offloading communication overhead, and optimizing such a computation–communication tradeoff is a large field of research in mobile edge computing (MEC) [119]–[122].

b) Memory limitation: The optimal NN model size for inference is determined by the devices' memory sizes of SRAM and DRAM. This may conflict with the optimal model size for the training operations, which is mainly determined by the communication overhead induced by the MSI exchanges. Furthermore, the limited size of the devices' storages, such as hard disk drive (HDD) and SSD, constrains the input data sample sizes as well as the number of intermediate calculation values stored during the training phase. Therefore, the training and inference operations need to be jointly optimized under the constraints of communication links, as well as of various types of devices' memories, respectively.

c) Privacy guarantees: Some of the data samples owned by the devices may be privacy-sensitive, e.g., medical records. Exchanging MSIs instead of data samples can partly preserve privacy, yet is still vulnerable to being reversely traced by eavesdroppers. Extra coding, such as homomorphic encryption [123], may guarantee privacy, but its processing delay may exceed the low-latency deadline. Exchanging redundant information may hide the private information but unfortunately results in extra communication delay. Inserting noise has a potential to protect privacy while playing a role as regularizer, as long as the noise level is properly adjusted; otherwise, it decreases the inference accuracy significantly.

2) Communication Bottlenecks:

a) Wireless capacity dynamics: In centralized ML, the communication links are implicitly assumed to be wired; for instance, PCI Express between controller-GPU connections or Ethernet between cloud-device connections under the master–device split. Compared to this, the communication links in edge ML are mostly capacity-limited wireless channels. Furthermore, the wireless channel capacity changes more frequently due to intermittent channel conditions and network congestion. To cope with this, the communication payload, i.e., MSI, needs to be properly compressed. With whom and how often to exchange the MSIs should be dynamically optimized.

b) Uplink–downlink asymmetry: Wireless cellular systems follow the helper–device split, where each edge BS becomes the helper. In this case, due to the device's lower transmission power, the uplink communication from the device to the helper is much slower than the downlink communication [124]. Such characteristics are not utilized in the aforementioned MSI exchanging methods in Section III-B2, where both uplink local MSI and downlink global MSI are of the same type with the identical payload size.

IV. THEORETICAL ENABLERS

In this section, we provide theoretical principles to characterize inference reliability and training dynamics in edge ML under the aforementioned communication and on-device constraints. Their related applications and techniques in the context of edge ML are exemplified in blue boxes.

A. ML Reliability Guarantees

Traditional ML focuses heavily on minimizing the average loss and/or average training latency. These approaches omit the credibility intervals of the calculations and therefore are insufficient for supporting URLLC applications. Instead, it is mandatory to evaluate the loss and latency that guarantee a target reliability. With this end, we first focus on the connection between inference reliability and training latency, followed by the relationship between communication latency and reliability during the training process for a given end application.

The training latency is determined primarily by the number of required training data samples, often referred to as sample complexity, until reaching a target inference accuracy with a target inference reliability. Unfortunately, traditional NN-based ML only outputs the target inference accuracy but not the target inference reliability. This missing block can be addressed by the Bayesian learning theory [125] and the probably approximately correct (PAC) framework [126].

1) Bayesian Learning: In contrast to traditional ML approaches, an NN is described not directly by the weights of the NN but by their stochastic distribution. The weight distributions before and after the training process are called as prior and posterior distributions, respectively. The prior distribution can be initialized by an arbitrary distribution or by accounting for the training data characteristics. For a given prior distribution, training an NN is recast as estimating the posterior distribution by maximizing a likelihood between the training data samples and the weights, as per the Bayes theorem

$$\underbrace{\Pr(w \mid \mathcal{D}_n)}_{\text{posterior}} = \underbrace{\Pr(w)}_{\text{prior}} \cdot \underbrace{\Pr(\mathcal{D}_n \mid w)}_{\text{likelihood}} / \Pr(\mathcal{D}_n) \tag{6}$$

where w is the set of NN's weights and $\mathcal{D}_n \subset \mathcal{D}$ is the set of training data samples. With the estimated posterior distribution, the interference with a set $\mathcal{D}' \subset \mathcal{D} \backslash \mathcal{D}_n$ of test data samples is described as

$$\underbrace{\Pr(y \mid \mathcal{D}')}_{\text{inference}} = \int \Pr(y \mid w, \mathcal{D}') \underbrace{\Pr(w \mid \mathcal{D}')}_{\text{posterior}} \, dw. \tag{7}$$

Calculating this by averaging over the posterior distribution is cumbersome. Instead, one can approximate the posterior distribution with its generated weights, e.g., via Markov chain Monte Carlo (MCMC) methods.

Bayesian Learning Techniques:

(a) *Gaussian Process (GP):* Gaussian-distributed priors with infinitely many perceptrons in a single hidden layer result in GP via the central limit theorem [127]. GP enables regression using the second-order statistics of Gaussian processes, thereby achieving low complexity in terms of sample and computation. GP has been used for controlling robots [128] and Google's Internet balloons [129].

(b) *Stochastic Gradient Langevin Dynamics (SGLD):* SGLD is the mini-batched Bayesian learning where the posterior distribution is approximated as the weights being generated via a form of MCMC [130], i.e., Langevin dynamics given as $\Delta w_k = (\eta_k/2)(\nabla \log \Pr(w_k) + (N/n)\sum_{i=1}^{n} \log \Pr(y_t \mid w_t) + \varepsilon_t$ for $\varepsilon_t \sim \mathcal{N}(0, \eta_t)$. This approximates SGD's weight update dynamics with an additional noise ε_t. Inserting ε_t is useful not only for reducing the generalization error, but also for connecting SGD with the differential privacy framework [131] (to be elaborated in Section IV-A and IV-C) and with Entropy SGD [97] (Section V-A).

2) PAC Framework: Inference uncertainty mainly comes from the unseen data samples at the training phase. To quantify this, for $n = |\mathcal{D}_n|$ training data samples, one can compare the trained NN's empirical average training loss, $\hat{L}(w) = (1/n)\sum_{i=1}^{n} \ell(y_i, w)$, with the expected inference loss, $L(w) = \mathsf{E}_\mu[\ell(y, w)]$, averaged over all seen/unseen data samples following a distribution μ. The PAC framework focuses on the bound of the difference between $L(w)$ and $\hat{L}(w)$, i.e., generalization error of the approximation–generalization tradeoff shown in Fig. 6(b), such that

$$\Pr\left(L(w) - \hat{L}(w) \leq \mathsf{GE}(\mu, \varepsilon)\right) \geq 1 - \varepsilon \tag{8}$$

where $\mathsf{GE}(\mu, \varepsilon)$ is the achievable generalization error with probability at least $1 - \varepsilon$. All terms in (8) depend on the NN's hypothesis $h \in \mathcal{H}$ with the entire hypothesis space $\mathcal{H}$ [132]. Treating an NN as an approximated function in regression, $\mathcal{H}$ implies the set of functions that the NN is allowed to select as being the solution [78]. When $\mathcal{H}$ is finite, applying Hoeffding's inequality and union bound, one can derive the GE as $\sqrt{(\log|\mathcal{H}| + \log(1/\varepsilon))/(2\,n)}$. Here, n is the sample complexity to satisfy a target generalization error with a target reliability $1 - \varepsilon$.

PAC Framework Applications:

(a) *PAC-VC Bound:* If the hypothesis space $\mathcal{H}$ is infinite, e.g., in deep NNs or non-parametric models such as GP, one can derive the GE using the Vapnik-Chervonenkis (VC) dimension $\mathsf{VC}(\mathcal{H})$. This yields the PAC-VC bound's GE $\sqrt{(\mathsf{VC}(\mathcal{H}) + \log(4/\varepsilon))/n}$. Here, $\mathsf{VC}(\mathcal{H})$ is simply calculated as the number of NN's parameters, and so is the PAC-VC bound.

(b) *PAC-Rademacher Bound:* The PAC-VC bound is quite loose in general, since its VC dimension is determined solely by the NN regardless of the training dataset. To rectify this, one can measure $\mathcal{H}$ using the Rademacher compexity $\mathsf{Rad}_{\mathcal{D}}(\mathcal{H})$ that depends not only on the NN model but also on the training dataset, yielding the GE of the PAC-Rademancher bound, given as $\mathsf{Rad}_{\mathcal{D}}(\mathcal{H}) + \sqrt{\log(1/\varepsilon)/n}$.

3) PAC-Bayesian Framework: In the previous PAC framework, PAC-VC bounds become accurate only in the case of as many training data as model parameters, which is infeasible particularly for deep NNs [133]. PAC-Rademancher bounds are free from the said limitation, yet become vacuous for modern NN architectures under ReLU activations with an SGD training process [134]. Utilizing Bayesian learning methods, one can resolve these issues, resulting in the following PAC-Bayes bound [135].

$$\Pr\left(L(q) - \hat{L}(q) \leq \sqrt{\frac{\mathrm{KL}(q||p) + \log(1/\varepsilon)}{2n}}\right) \geq 1 - \varepsilon \tag{9}$$

where $L(q)$ and $L(\hat{q})$ denote the empirical and expected average loss values for a posterior q, respectively. In (9), the GE is a function of the Kullback–Leibler (KL) divergence of q and a prior p, which is also called as the complexity required for mapping p to q. It is worth noting that the difference between $L(q)$ and $\hat{L}(q)$ can be measured using their KL divergence, yielding the refined version in

[136] as stated in the following:

$$\Pr\left(\mathrm{KL}\left(\hat{L}(q)\|L(q)\right) \leq \frac{1}{n}\left[\mathrm{KL}(q\|p)+\log\frac{n+1}{\varepsilon}\right]\right) \geq 1-\varepsilon. \tag{10}$$

Both (9) and (10) are derived under an IID training data set, which can be extended to a non-IID data set as described in the following applications.

PAC-Bayes Framework Applications:

(a) *Chromatic PAC-Bayes Bound for Non-IID Data:* When the dependency in a training dataset $\mathcal{D}_n$ is modeled as a dependency matrix $\Gamma(\mathcal{D}_n)$ using fractional covers in graph theory, the PAC-Bayes bound becomes

$$\Pr\left(\mathrm{KL}\left(\hat{L}(q)||L(q)\right) \leq \frac{\chi^*}{n}\left[\mathrm{KL}(q||p)+\log\frac{n/\chi^*+1}{\varepsilon}\right]\right) \geq 1-\varepsilon$$

where χ^* is the fractional chromatic number of the dependency matrix $\Gamma(\mathcal{D}_n)$, defined as the minimum weight over all proper exact fractional covers of all vertices in $\Gamma(\mathcal{D}_n)$. The fractional chromatic number can be obtained using linear programming, and its closed form is available if $\Gamma(\mathcal{D}_n)$ belongs to some special classes of graphs [137]. The definition of $\Gamma(\mathcal{D}_n)$ is detailed in [138].

(b) *Collective Stable PAC-Bayes Bound for Non-IID Data:* Consider $\mathcal{D}_n$ is divided into m subsets. When the dependency among m subsets is modeled using the collective stability framework, the PAC-Bayes bound is given as

$$\Pr\left(L(q) - \hat{L}(q) \leq 2\beta||\Gamma||_\infty\sqrt{\frac{\mathrm{KL}(q||p) + \log(2/\varepsilon)}{2nm}}\right) \geq 1-\varepsilon$$

where β is the Lipschitz constant under the Hamming distance between the training inputs. The term Γ is the training dataset's dependency matrix whose definition is elaborated in [139].

(c) *PAC-Bayes Bound With Data-Dependent Priors:* The Bayesian prior p of a PAC-Bayes bound should be chosen independently of the training dataset $\mathcal{D}_n$, yet can still depend on the distribution of the dataset. Utilizing this idea, it is possible to characterize the dependency between $\mathcal{D}_n$ and p via the differential privacy framework [140], yielding the following PAC-Bayes bound

$$\Pr\left(\mathrm{KL}\left(\hat{L}(q)||L(q)\right) \leq \frac{1}{n}\left[\mathrm{KL}(q||p)+2c(\varepsilon,\epsilon)\right]\right) \geq 1-\varepsilon$$

where $c(\varepsilon,\epsilon) = \max\{\log(3/\varepsilon), n\epsilon^2\}$, and ϵ implies that the data-dependent prior p ensures ϵ-differential privacy whose definition is elaborated in Section IV-B.

4) Meta Distribution: In both PAC and PAC-Bayes frameworks, the obtained generalization error bounds hold for any n number of selected training samples, as they are averaged over the selections of the training data set. This is suitable for centralized ML where the training minibatched data samples are frequently renewed. For edge ML, such a scenario may not be feasible due to the communication overhead and/or privacy guarantees. In this case, the meta distribution enables to capture the generalization error bound, by accounting for a given set $\mathcal{D}_n$ of the training samples as follows:

$$\Pr\left\{\Pr\left(L(\mathcal{D})-\hat{L}(\mathcal{D}_n)\leq GE(\mu_n,\varepsilon)\mid \mathcal{D}_n\right)\geq 1-\varepsilon\right\}\geq 1-\delta \tag{11}$$

where μ_n is the distribution of the data samples in $\mathcal{D}_n$ and δ is the target generalization error outage probability for all training samples. In order not to complicate the calculation, one can approximate the meta distribution with the beta distribution, as proposed in [141]. This is the second-order moment approximation and thus only requires to calculate the mean and variance of the inner-most probability in (11).

Meta Distribution Applications:

(a) *Signal-to-Interference-Ratio (SIR) Meta Distribution:* The idea of meta distributions was originally proposed in the context of stochastic geometry [141]. It focuses on the meta distribution of the SIR coverage probability for *a given large-scale network topology*. One can thereby quantify, for instance, the fraction of receivers that guarantees a target wireless communication reliability, which in passing is also useful for the latency reliability analysis in large-scale edge ML design.

(b) *Probably Correct Reliability (PCR):* Focusing on estimating wireless communication channels, PCR is the meta distribution of outage capacity for *a given set of channel observations* [142]. This reliability analysis framework is promising in the context of MLC towards enabling URLLC.

5) Risk Management Framework: The above-mentioned reliability bounds are determined by the difference between the biased loss after training and the ideally averaged loss with the entire data samples, i.e., generalization error. Instead, one may inquire whether the biased loss reliably achieves a target loss level or not. The right tail of the biased loss distribution can describe this, which has been investigated in mathematical finance using value-at-risk (VaR) and conditional-value-at-risk (CVaR) [143]. VaR focuses on the tail's starting point, hereafter referred to as the tail threshold, specifying the minimum target loss x

that guarantees a target reliability $1-\varepsilon$ as follows:

$$\mathrm{VaR}_{1-\varepsilon}(\hat{L}) = \arg\min_{x} \left\{ \Pr\left(\hat{L} \leq x\right) \geq 1-\varepsilon \right\}. \tag{12}$$

VaR is often defined by a nonconvex and/or discontinuous loss function, which requires complicated calculations. Even with such a loss function, CVaR avoids this complication, by ensuring its monotonicity. To this end, CVaR considers the tail's area, providing the expectation of the loss exceedances with the tail threshold set as the VaR

$$\mathrm{CVaR}_{1-\varepsilon}(\hat{L}) = \mathsf{E}\left[\hat{L} \mid \hat{L} > \mathrm{VaR}_{1-\varepsilon}(\hat{L})\right]. \tag{13}$$

Risk Management Framework Applications:

(a) *Distributional RL:* Traditional RL is trained so as to approximate the expected cumulative return at the current state x and action a, e.g., Q-function $Q(x,a) = \mathsf{E}[R(x,a) + \gamma Q(x',a')]$ for the reward $R(x,a)$ and a discount factor γ when $(x,a) \rightarrow (x',a')$. Instead, one can learn to approximate the distribution of $Z = R(x,a) + \gamma Q(x',a')$, known as the value distribution. Such distributional RL outperforms state-of-the-art RL methods including deep Q-learning and asynchronous actor-critic RL [144], [145].

(b) *Risk-Sensitive RL:* Maximizing the expected return in RL is insufficient for robust control warranting a predefined minimum return with a target probability. The mean-variance approach is the simplest solution that additionally minimizes the variance of returns while maximizing the expected return [146]. A better approach is to maximize the $(1-\varepsilon)$-worst return, i.e., CVAR, by observing extra samples as done in [147]. Furthermore, one can utilize the value distribution of distributional RL, thereby directly computing and maximizing CVaR [148].

6) Extreme Value Theory: The right tail of the biased loss distribution can also be described in the context of two fundamental theorems in an extreme value theory (EVT) [149]. To elaborate, for any distribution of the biased loss, the Pickands–Balkema–de Haan theorem states that the distribution of its loss exceedances with the infinitely large tail threshold θ converges to the generalized Pareto distribution (GPD), that is

$$\Pr(\theta < \hat{L}(\mathcal{D}_n) \leq x) \overset{\theta \rightarrow \infty}{=} \underbrace{1-(1+\zeta x/\sigma)^{-1/\zeta}}_{\mathrm{GPD}(x,\zeta)} \tag{14}$$

which becomes a Pareto (if $\zeta < 0$), an exponential ($\zeta = 0$), or a uniform distribution ($\zeta = -1$). Next, focusing on the training data set $\mathcal{D}_n \subset \mathcal{D}$, one may need to guarantee the reliability even for the worst case loss with respect to a number K of the IID training data set selections $\{\hat{L}(\mathcal{D}_{n,k})\}_{k\leq K}$ in centralized ML or to K devices in edge ML. In this case, the Fisher–Tippett–Gnedenko theorem of EVT is applicable. The theorem describes that the distribution of the maximum loss out of the loss values obtained by an infinitely large number of the training data set selections converges to the generalized extreme value (GEV) distribution, that is

$$\Pr\left(\hat{L}_K(\mathcal{D}_n) \leq x\right) \overset{K\rightarrow\infty}{=} \underbrace{e^{\mathrm{GPD}(x-m,\zeta)-1}}_{\mathrm{GEV}(x,\zeta)} \tag{15}$$

where $\hat{L}_K(\mathcal{D}_n) = \max\{\hat{L}(\mathcal{D}_{n,k})\}_{k\leq K}$ and m is the mean of $\{\hat{L}(\mathcal{D}_{n,k})\}_{k\leq K}$. The GEV becomes a Gumbel (if $\zeta = 0$), a Fréchet ($\zeta > 0$), or a reversed Weibull distribution ($\zeta < 0$). The relationship between GEV and GPD is obtained trivially by applying Taylor's expansion to (15) for $\theta < x \rightarrow \infty$.

EVT Applications:

(a) *Robust Multiclass Classification:* Each data sample in multiclass classification has one ground-truth label out of multiple labels. In this case, the robustness is defined as the maximum noise (both in attributes and labels) that prevents adversarial inputs. To evaluate the upper bound for the noise, the maximal gradients of the classifier functions for each label need to be evaluated over all training samples. Alternatively, using EVT, one can evaluate the maximum gradients using only a few training samples [150].

(b) *Kernel-Free Open-Set Recognition:* Supervised classification in open sets with unknown number of classes has a challenge of classifying inputs belonging to unseen classes at training time due to under sampling. The solution is a kernel-free recognition technique known as *extreme value machine* [151]. Therein, EVT is used to characterize the decision boundaries of classes in a probabilistic manner that provide an accurate and efficient data partition.

7) Optimal Transport Theory: Minimizing the KL divergence $\mathrm{KL}(p_Y||p_X)$ between two distributions p_X and p_Y is ubiquitous in the NN training and inference processes. For training, it is identical to maximizing the likelihood in Bayesian learning. It also captures training a generative model such as GAN whose loss function is defined using the KL divergence. For inference, it minimizes the variance term in the PAC-Bayesian bound. The caveat is that the KL divergence becomes infinite if two distributions have nonoverlapping supports, hindering the reliability of training and inference processes. To resolve this, as shown in Fig. 11, a naïve approach is to insert noise into the distributions so as to secure a common range of support, which, however, compromises accuracy. Instead, replacing the KL divergence minimization with calculating the Wasserstein distance in optimal transport (OT) [152] has recently become a promising solution, which yields a finite value even with nonoverlapping distributions. The simplest case is calculating the Wasserstein-1 distance $W(p_Y||p_X)$,

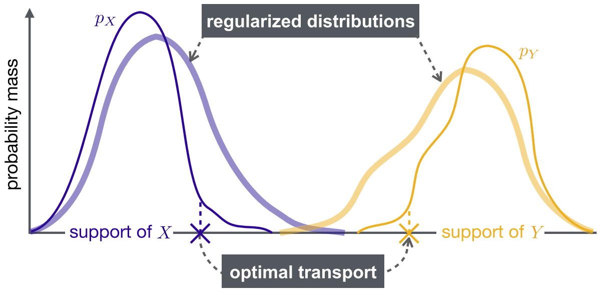


Fig. 11. ***OT, compared to regularized distributions.***

given as

$$W(p_Y||p_X) = \inf_{\gamma \in \Pi_{X,Y}} \mathsf{E}_{X,Y\sim\gamma}\left[||X-Y||\right]. \tag{16}$$

This implies the minimum cost to transport the probability mass from X to Y so that p_X equals to p_Y, as shown in Fig. 11. The transported mass amount is characterized by the joint distribution γ of X and Y. The constraint $\gamma \in \Pi_{X,Y}$ ensures that such an optimal transportation γ^* has its marginal distributions p_X and p_Y. The calculation of $W(p_X||p_Y)$ by deriving γ^* is challenging, thus commonly relying on approximation and algorithmic techniques.

OT Applications:

(a) *Wasserstein GAN (WGAN):* The training process of WGAN [80] is equivalent to calculating the Wasserstein distance. For its easier calculation, the RHS of (16) is replaced with its dual formulation $\sup_{||f||_L\leq 1} \mathsf{E}_{X\sim P_X}[f(X)] + \mathsf{E}_{Y\sim P_Y}[f(Y)]$ by applying the Kantorovich-Rubinstein duality. This implies a regression problem with a 1-Lipshitz function f, which can be solved via supervised learning.

(b) *Sinkhorn Divergence:* Instead of the said dual approach, [153] tackles the primal formulation in (16) with an entropic approximation and the Sinkhorn algorithm [154]. In this approach, the RHS of (16) is recast as $\inf_{M\in\mathcal{M}} \mathrm{Tr}(MC^T)$ with the cost matrix C. This is a matching problem of the matrix M given by mini-batched X and Y, which can be solved via the parallelized Sinkhorn algorithm.

8) Rényi Entropy: The IB in Section III-A2 describes the information flow across the consecutive L layers of an NN during the training process. This can be achieved by extending a single bottleneck $\hat{X}$ in (2) into L bottlenecks $\{X_l\}$ values [89]. When the bottlenecks are discrete random variables, the flow dynamics are described by the discrete Shannon entropy [155]. By contrast, when the bottlenecks are continuous random variables, the dynamics are expressed using continuous entropy, i.e., differential entropy, which unfortunately results in intractable solutions, except when the input X and the output Y are jointly Gaussian [156]. In order to derive a tractable solution, one can leverage the Rényi entropy

$$H_\alpha(X) = \log_2\left(\sum_{i=1}^{n} {p_i}^{\alpha}\right)/(1-\alpha) \tag{17}$$

where its limiting case becomes the Shannon entropy $H_1(X) = -\sum_{i=1}^{n} p_i \log_2\ p_i$ when α approaches 1. With its matrix version expression, the IB formulation with L layers is given as

$$\min_f \sum_{i=1}^{L} I_\alpha(X;X^l) + \beta H(Y, f(X)) \tag{18}$$

where f is a function that the NN tries to learn and $H(Y, f(X))$ is the cross entropy between Y and $f(X)$. The term $I_\alpha(X;X^l)$ is the matrix-based Rényi mutual information that equals $I_\alpha(X;X^l) = S_\alpha(X) + S_\alpha(X^l) - S_\alpha(X,X^l)$. Here, the first term is the matrix-based Rényi entropy that equals $S_\alpha(X) = \log_2 \mathrm{Tr}(X^\alpha)/(1-\alpha)$. The last term is the matrix-based joint Rényi entropy that is given by $S_\alpha(X,X^l) = S_\alpha(X \circ X^l/\mathrm{Tr}(X \circ X^l))$, where $\circ$ is the Hadamard product.

B. Latency Reduction and Scalability Enhancement

The performance of data-driven ML approaches rests on how many data samples are utilized. In edge ML in which data samples are generated by the devices, the problem boils down to how many devices can be federated. Providing a privacy-preserving mechanism during their federation is key to increasing the range of federation, which can be addressed through the lens of DP [157], [158].

A large range of federation brings to the fore the subsequent question, how to cope with the MSI communication costs of a large number of edge ML devices whose NN model sizes may exceed the wireless channel capacity. In this respect, a rate–distortion theory establishes a guideline for compressing MSI by balancing the compression rate and its resulting distortion [91], [159]. Furthermore, in a multiagent RL (MARL) setting, mean-field game (MFG) theory [160], [161] provides an elegant method making full use of local computations, as detailed next.

1) Differential Privacy: With local data sets owned by devices, distributed training operations should preserve the data privacy. To this end, one can apply an auxiliary mechanism, such that its output cannot tell whether a particular local data set is participated in the training operations, thereby preserving data privacy. DP formalizes this idea while quantifying the privacy loss of a mechanism $\mathcal{M}(\cdot)$ and its target threshold ϵ as follows:

$$\underbrace{\log\left(\frac{\Pr\left(\mathcal{M}(\mathcal{D}_i)\in\mathcal{S}\right)}{\Pr\left(\mathcal{M}(\mathcal{D}_j)\in\mathcal{S}\right)}\right)}_{\text{privacy loss}} \leq \epsilon \tag{19}$$

which holds for any subsets $\mathcal{D}_i$ and $\mathcal{D}_j$ with $i \neq j$ out of the entire data set $\mathcal{D}$. When the aforementioned constraint is satisfied, the mechanism achieves ϵ-DP, in which an adversary can only differentiate whether the output is from $\mathcal{D}_i$ or $\mathcal{D}_j$, with an uncertainty inversely proportional to ϵ. For SGD, this is achieved by simply inserting a Gaussian noise into each data input, i.e., the Gaussian mechanism [158]. For Bayesian learning, one can sample the weights from the noise inserted posterior distribution, i.e., the exponential mechanism [157], achieving the ϵ-DP.

DP Application:

(a) *DP Regularizer:* The cost for achieving ϵ-differential privacy is the increase in noise. This is not always a foe but can be a friend. In fact, inserting noise is treated as adding a regularizer. Thus, so long as the noise level is appropriately adjusted, one can improve both accuracy and privacy, as done in [162].

2) Rate–Distortion Theory: Quantizing MSI reduces latency by decreasing the communication payload size, at the cost of compromising MSI accuracy due to the distortion induced by quantization. Balancing latency and accuracy can, therefore, be achieved by optimizing the number ℓ of quantization levels. The rate–distortion theory characterizes the optimum ℓ under a given channel condition, by stating that the transmitting rate $R(D)$ bits/sample with distortion D should not exceed a given wireless channel capacity C bits/sample, i.e., $R(D) \leq C$. To elaborate, when the MSI is treated as an IID Gaussian source with variance σ^2, the Shannon distortion–rate function [91] is given as

$$R(D) = 1/2 \cdot \log_2(\sigma^2/D). \tag{20}$$

Using a uniform scalar quantizer with a sufficiently small step size Δ, the mean squared error distortion is approximated as $D \approx \Delta^2/12$ [159]. Next, assuming that the source deviation σ equals the difference between the maximum and minimum quantized values, we obtain the quantization levels $\ell = \sigma/\Delta$. Applying these two results to (20) thereby leads to the following bound:

$$C \geq R(D) \approx \log_2(\ell) + \log_2(12)/2. \tag{21}$$

This shows the upper bound of quantization levels, which is useful when transmitting the MSI as much as possible for a given channel condition. Since the Gaussian source assumption yields the lowest rate, the result provides the worst case quantizer design guideline in practice.

Rate-Distortion Theory Applications:

(a) *Regularization via Quantization:* If one is willing to minimize the MSI communication latency, the lower bound of quantization levels is needed. In fact the distortion can contribute positively to the regularization in edge ML [163]. The MSI can thus be distorted until reaching an optimal regularizing noise level. This maximum distortion yields the lower bound.

(b) *IB Under KL Divergence:* The formulation (2) in IB is a special case for obtaining the rate-distortion function $R(D)$ with a distortion measure given as a KL divergence [164]. Namely, (2) originates from $R(D) = \min_{p(\hat{x};x)} I(X;\hat{X})$ s.t. $\sum_{x,\hat{x}} p(x)p(\hat{x}|x)\mathcal{D}(x|\hat{x})D$, where $\mathcal{D}(x|\hat{x})$ is a distortion measure between the original input x and its compressed information $\hat{x}$. Then, its Lagrangian relaxed formulation with the distortion measure set as the KL divergence between $p(y|x)$ and $p(y|\hat{x})$ becomes (2).

3) Mean-Field Control Framework: In MARL, N devices are strategically interacting by individually taking actions without a central coordinator, formulated as an N-player game. In this game, both computational complexity and the number of communication rounds across devices increase exponentially with N, which can be remedied at using the MFG theory [160].

To illustrate, consider a three-player game with devices $A, B, C \in \mathcal{P}$. Device A with a given state first takes an action that affects the states of the other devices, i.e., device A interacting with $\mathcal{P}\backslash A$. Likewise, device B interacts with $\mathcal{P}\backslash B$, while device C interacts with $\mathcal{P}\backslash C$. All these problems are coupled, and calculating their Nash equilibrium induces the undesirable complexity. However, if N becomes extremely large, in the aforementioned example, it becomes $\mathcal{P}\backslash A \approx \mathcal{P}\backslash B \approx \mathcal{P}\backslash C \approx \mathcal{P}$, since each action is likely to affect a negligibly small fraction of the entire population. This implies that each device plays a two-player game with a virtual device, i.e., the entire population, which is called an MFG. As illustrated in Fig. 12, the originally coupled N-player game thereby becomes a number N of individual device's two-player games that can be locally solved for a given state of the entire population, referred to as the MF distribution. The MF distribution is obtained by solving the Fokker–Planck–Kolmogorov (FPK) equation of a continuous Markov process [161]. For the given MF distribution, the optimal action of each device is then taken by solving the Hamilton–Jacobi–Bellman (HJB) equation, a continuous version of the Bellman backward equation in MDP [161]. An MFG theoretic communication-efficient UAV control example will be elaborated in Section VI-G.

MF Control Framework Applications:

(a) *MFG in Wireless Systems:* With the aforementioned wind dynamics, the work [165] investigated the UAV mobility control that avoids inter-UAV collisions while maximizing their air-to-ground communication performance. With the wireless channel dynamics, MFG has shown its effectiveness in transmission power control and resource management particularly under ultra-dense cellular networks (UDNs) [166]. Moreover, the spatio-temporal content popularity dynamics were modeled in [167], and an optimal file caching strategy was found using MFG.

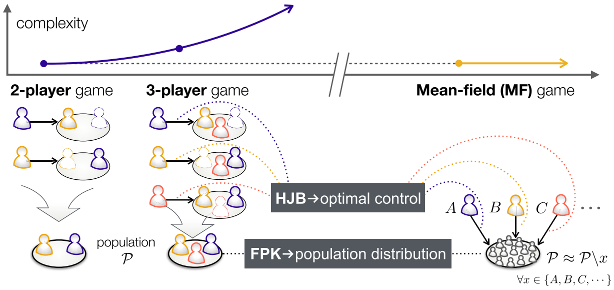


Fig. 12. ***MFG where each device locally plays a two-player game with a virtual device, i.e., the entire population whose state is given by the MF distribution.***

(b) *MF Control in ML:* With the feed forward dynamics of the deep NN training process, the optimal weights are determined by an HJB that can be solved by the Euler approximation method [168], to be detailed in Section V-D2. Note that all the said examples assume that the initial state of the population is given by a Gaussian distribution that is not always realistic. To fill this gap, in the context of large-scale MARL, the population dynamics of the MFG has been inferred using an inverse RL method in [169].

V. TECHNICAL ENABLERS

In this section, we propose technical solutions that enable low-latency decentralized training as well as reliable and accurate decentralized inference under communication and on-device constraints. Their relationships with the theoretical principles presented in Section III are elaborated upon in yellow boxes.

A. ML Reliability Improvement

Generalization errors can be reduced by designing an NN training algorithm for finding flat minima in the EL, as exemplified by entropy SGD [97]. For multiple different tasks, one can reduce the generalization errors by training NNs based on task correlations [170], [171]. On another level, the training process can become robust against malicious and/or malfunctioning devices by the aid of blockchain technologies [172], as elaborated next.

1) Entropy SGD: The goal of entropy SGD is to obtain a flat minimum solution [97]. To this end, for a given original loss function $\mathcal{L}(w)$, entropy SGD minimizes its modified loss function, referred to as local entropy, given as

$$u(w_k, \gamma) = -\log \mathsf{E}_g \left[e^{-(\mathcal{L}(w_k)+g)} \right] \tag{22}$$

where g follows a Gaussian distribution with variance γ. The local entropy loss is designed by Gaussian sampling from the $\mathcal{L}(w)$'s Gibbs entropy that is proportional to the number of local maxima within γ in the EL. The local entropy can be minimized using an MCMC algorithm [173].

Entropy SGD Related Theories:

(a) *Serial-to-Parallel Conversion via SGLD:* It is remarkable that entropy SGD for a single device is identical to elastic SGD operated with multiple devices, under an ergodicity condition $\nabla^2\ L(w) + 1/\gamma I\ \succ\ 0$. Edge ML can hence be analyzed by reusing most of the theoretical principles that were originally applicable for a single device. The said conversion is validated by first exploiting the Hope-Cole transformation [97], [173] that shows the local entropy loss of entropy SGD is the solution of a viscous Hamilton–Jacobi partial differential equation (PDE). Solving this PDE using a homogenization technique [97], [173] yields the loss dynamics that is identical to the elastic SGD's loss dynamics.

(b) *PAC-Bayes Bound for Entropy SGD:* A recent work [131] verifies that entropy SGD works by optimizing the Bayesian prior, i.e. the distribution of $w_k + g$. This clarifies the difficulty of deriving the PAC-Bayes bound for entropy SGD, as the prior of the original PAC-Bayes framework is constrained to be chosen independently of the training data. The work [131] resolves this problem, by first adding an extra random noise to entropy SGD's weight updates, as done in SGLD [130]. The resulting algorithm is referred to as entropy SGLD that is interpreted as a mechanism achieving ϵ-differential privacy. Then, utilizing the PAC-Bayes with the data-dependent prior satisfying the ϵ-differentially privacy (see Section IV-A) yields the PAC-Bayes bound for entropy SGD, thereby quantifying its generalization capability.

2) Task-Aware Training: Generalization errors result not only from unseen dispersed samples (see Section IV-A) but also from distinct tasks of devices. For a given set of tasks, multitask learning (MTL) [170], [174] trains an NN so as to ensure high accuracy for multiple tasks by inserting a task correlation regularizer into the original loss function [174]. When the task correlation is unknown, the correlation can also be trained by alternating: 1) optimizing the NN weights while fixing the correlation matrix and 2) optimizing the correlation matrix while fixing the NN weights [170]. On the other hand, if the tasks are not given *a priori*, MTL is ill-suited because task correlations cannot be specified. Alternatively, meta learning is still effective in this case, which aims to train a meta-learner that is capable of quickly learning various types of tasks [171]. This training objective can be achieved by randomly sampling the loss function out of all possible loss functions corresponding to different known tasks, thereby guaranteeing robustness against unseen tasks.

3) Block-Chained Training: In edge ML, malicious devices may participate and disrupt the training process. Furthermore, selfish devices may not contribute to the local training procedures while only receiving the global

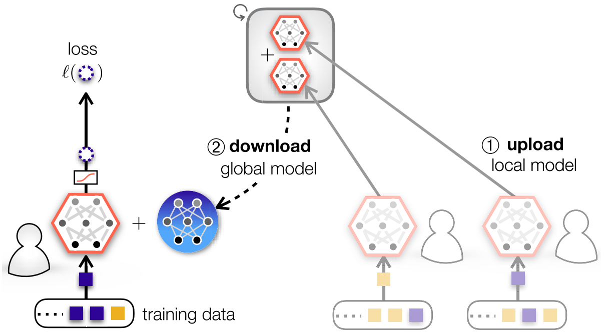


Fig. 13. ***Operational structure of FL.***

training results computed by the other devices. Keeping a record of the training process using distributed ledger technology (DLT) is useful to mitigate these problems. For example, when exchanging local MSIs, each device cross-validates the MSIs and stores the accepted MSIs in its local distributed ledger. The locally distributed ledger is synchronized with the other devices' ledgers via DLT, such as a blockchain algorithm [172] or a directed acyclic graph (DAG)-based Byzantine fault tolerance (BFT) algorithm [175], [176]. Thereby, only the legitimate local MSIs contribute to the global MSI calculation, ensuring the reliability of the training process.

B. Communication-Aware Latency Reduction

In what follows, we introduce and propose MSI exchange schemes to reduce the communication latency during distributed training operations. We hereafter focus on the h-d architectural split where training data samples are generated by devices unless otherwise specified.

1) Periodic Model MSI Exchange:

a) Federated averaging: Too frequent MSI exchanges in CSGD incur significant communication overhead. In order to mitigate this, the devices in federated averaging (FAvg) exchange the local MSIs at an interval of τ epochs, as illustrated in Fig. 13 [28]. Following the notations defined in Section III-B2, at the kth epoch, the ith device's weight $w_k^{(i)}$ is described as follows:

$$w_{k+1}^{(i)} = \begin{cases} w_k^{(i)} - \eta\bar{g}_k, & \text{if } k \bmod \tau = 0 \\ w_k^{(i)} - \eta g\big(w_k^{(i)}\big), & \text{otherwise.} \end{cases} \tag{23}$$

Similar to CSGD, the learning rate η needs to decrease with k in order to reduce the weight update variance induced by the randomness of the user-generated training samples.

b) Federated SVRG: Allowing a constant learning rate can reduce the training latency compared to FAvg whose learning rate decreases with time. As a variant of FAvg, federated stochastic variance reduced gradient (FSVRG) applies the SVRG [177] that minimizes the weight update variance by additionally utilizing the difference between the local and global gradients, allowing a constant learning rate [30]. Besides, similar to ESGD, FSVRG keeps track of the global MSI and updates it based on the distributed approximate Newton (DANE) [178] that also ensures a constant learning rate. This yields the following local and global weight update rules:

$$w_{k+1}^{(i)} = \begin{cases} w_k^{(i)} - \eta\big(\bar{g}(\hat{w}_k) + g\big(w_k^{(i)}\big) - g(\hat{w}_k)\big), & \text{if } k \bmod \tau = 0 \\ w_k^{(i)} - \eta g\big(w_k^{(i)}\big), & \text{otherwise} \end{cases} \tag{24}$$

where

$$\hat{w}_k = \hat{w}_{k-1} + \sum_{i=1}^{M} \frac{n_i}{n}\big(w_k^{(i)} - \hat{w}_{k-1}\big), \quad \text{if } k \bmod \tau = 0.$$

$\bar{g}(\hat{w}_k) = 1/M \sum_{i=1}^{M} g(\hat{w}_k^{(i)})$. The notation n_i is the local training data set size, and n is the size of the entire devices' aggregate data set, i.e., global data set. Like FAvg, FSVRG applies the periodic MSI exchanges at an interval of τ, reducing the communication overhead. Nonetheless, FSVRG requires extra MSI weight exchanges, in addition to the gradients of FAvg. Thus, the resulting communication payload size is doubled from FAvg.

Both FAvg and FSVRG are often referred to as FL that works under non-IID training data sets in practice [28], [30], though the accuracy is degraded compared to the best case performance under the IID data sets [179], [180].

c) Codistillation: While keeping communication overhead the same as in FAvg, codistillation (CD) has the potential to obtain a more accurate model by exploiting extra computation and memory resources during the training phase [181]. In fact, all the aforementioned training methods directly apply the globally averaged MSI to the local MSI update calculation via an ensembling method. Alternatively, one may focus on the fact that after downloading the global average weight MSI evaluated by exchanging local weight MSI, each device can have two separate NN models: its local model and the globally averaged model.[1] As shown in Fig. 14, in the next local weight update, the device feeds a training data sample to both models. Then, from the KD point of view, the global model is interpreted as a teacher whose inference output, i.e., knowledge, can be transferred to a student, i.e., the local model. This is enabled by the following weight update rule:

$$w_{k+1}^{(i)} = \begin{cases} w_k^{(i)} - \eta\big(g\big(w_k^{(i)}\big) + \psi\big(F_k^{(i)}, \hat{F}_k^{(i)}\big)\big), & \text{if } k \bmod \tau = 0 \\ w_k^{(i)} - \eta g\big(w_k^{(i)}\big), & \text{otherwise.} \end{cases} \tag{25}$$

[1] The original CD in [181] considers that each device has: 1) its local model and 2) all copies of the other devices' models, which may incur huge communication overhead. Instead, we replace 2) with the globally averaged model, without loss of generality.

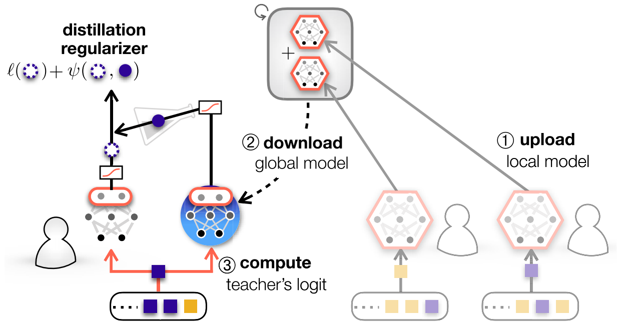


Fig. 14. ***Operational structure of CD.***

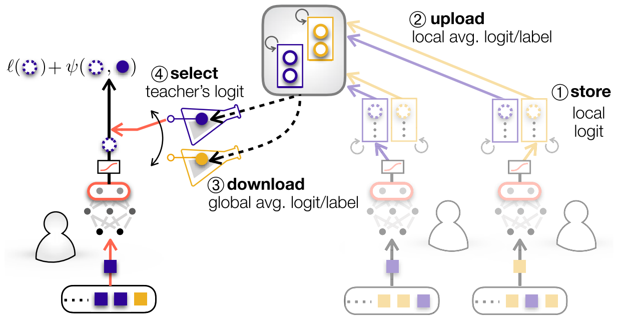


Fig. 15. ***Operational structure of FD.***

The local model's term $F_k^{(i)}$ is a set of normalized logits using a modified softmax function with a temperature hyperparameter T, henceforth denoted as a logit vector. To illustrate, for the ℓth label's logit x_ℓ with L labels, the normalized value equals softmax$(x_\ell) = \exp(x_\ell/T)/\sum_{j=1}^{L}\exp(x_j/T)$. Adjusting the temperature T helps the logits to be closely mapped into the per-label inference probabilities, such that the normalized values become identical across all labels or the maximum logit for $T \to \infty$ or 0, respectively. Likewise, the global model's logit vector is given as $\hat{F}^{(i)}$. With these local and global logit vectors, the distillation regularizer $\psi(F_k^{(i)}, \hat{F}_k^{(i)})$ is given as the gradient of their mean squared error [182] or of the cross entropy [118], [181]. As such, when the local model's output $F_k^{(i)}$ is close to the global model's output $\hat{F}_k^{(i)}$, i.e., small distillation regularizer, the local weight is determined mostly by the locally calculated gradient and otherwise perturbed in order not to follow the local bias. In CD, due to the periodic communication interval τ, also known as a checkpoint interval, the communication overhead becomes as small as FAvg. Its downside is consuming extra memory and computation resources for separately storing the global model and performing inference using the global model.

2) Output MSI Exchange:

a) Federated distillation: Exchanging the model parameter MSI, i.e., weights and/or gradients, may induce large communication payload size particularly for a deep NN, since the number of parameters is proportional to the model size. To resolve this, we propose federated distillation (FD) that exchanges the model output MSI, i.e., normalized logits whose payload size depends only on the output dimension, i.e., the number of labels [180]. The weight update rule is then implemented using KD. The key challenge is that a set of normalized logits, henceforth denoted as a logit vector, is associated with its input training data sample. Therefore, to operate KD between the exchanged global average logit vector and the local model's logit, both logit vectors should be evaluated using an identical training data sample. Unfortunately, synchronous logit vector exchanges as many as the training data set size brings about significant memory and communication costs. Such communication cost may even exceed the model parameter MSI payload, which is the reason why CD resorts to parameter MSI exchanges.

To rectify this, as illustrated in Fig. 15, each device in FD exchanges a set of mean logits per label, each of which is locally averaged over epochs until a checkpoint. This local average logit MSI is associated not with individual data samples but with the accumulated training data set, enabling periodic local MSI exchanges. The exchanged local MSIs are then averaged across devices, yielding a set of global mean logits per label, i.e., global average logit MSI. At the next local weight update phase, each device selects distillation regularizers that are synchronous with its training data samples. This weight update rule is represented as follows:

$$w_{k+1}^{(i)} = \begin{cases} w_k^{(i)} - \eta\big(g(w_k^{(i)}) + \psi(\bar{F}_{k,\ell}^{(i)}, \breve{F}_{k,\ell}^{(i)})\big), & \text{if } k \bmod \tau = 0 \\ w_k^{(i)} - \eta g(w_k^{(i)}), & \text{otherwise} \end{cases} \tag{26}$$

where $\bar{F}_{k,\ell}^{(i)}$ is the local average logit vector when the training sample belongs to the ℓth ground-truth label. The global average logit vector equals $\breve{F}_{k,\ell}^{(i)} = \sum_{j\neq i}\bar{F}_k^{(j)}/(M-1)$. The exchanged local average logit MSI is the set of per-label local average logit vector for all labels, i.e., $\{\bar{F}_{k,\ell}^{(i)}\}_{\ell=1}^{L}$, so does the global average logit MSI $\{\breve{F}_{k,\ell}^{(i)}\}_{\ell=1}^{L}$.

The performance of FD can further be improved with a slight modification. In fact, model output accuracy increases as training progresses. Thus, it is better to use a local weighted average logit MSI, where the weight increases with time. Alternatively, one can implement FD by only exchanging the most mature knowledge. For instance, consider that devices share a knowledge test data set *a priori*. Just before each checkpoint, using the test set, each device measures its latest inference knowledge. Then, FD is enabled by exchanging local checkpoint logit MSI, a set of local logit vectors obtained by the test data set, each of which corresponds to a single ground-truth label, exchanging the local MSI.

b) Federated Jacobian distillation: Model input–output Jacobian matching is interpreted as a KD operation that inserts noise into logits [185]. Since noisy logit exchanges improve the KD performance when the noise level is properly adjusted [163], we expect exchanging Jacobian matrices to improve FD. With this motivation, the weight update rule of FJD is given as

$$w_{k+1}^{(i)} = \begin{cases} w_k^{(i)} - \eta\big(g(w_k^{(i)}) + \psi(\bar{G}_{k,\ell}^{(i)}, \breve{G}_{k,\ell}^{(i)})\big), & \text{if } k \bmod \tau = 0 \\ w_k^{(i)} - \eta g(w_k^{(i)}), & \text{otherwise.} \end{cases} \tag{27}$$

In the distillation regularizer, $\bar{G}_{k,\ell}^{(i)}$ is the local average of the logit vector's Jacobian when the input sample's ground-truth belongs to the ℓth label and $\breve{G}_{k,\ell}^{(i)}$ is the global average of local average logit Jacobian vectors for the ℓth label. Following FD, the local and global MSIs are given as $\{\bar{G}_{k,\ell}^{(i)}\}_{\ell=1}^{L}$ and $\{\breve{G}_{k,\ell}^{(i)}\}_{\ell=1}^{L}$, respectively. Compared to FD, the disadvantage is that FJD requires extra memory and computation resources for storing and computing the Jacobian matrices. Another burden is that Jacobian matrix communication leads to the payload size being proportional to the output dimension multiplied by the input dimension, which is nonetheless still independent of the model size.

3) Uplink–Downlink Asymmetric MSI Exchange: Due to the device-limited transmission power, the uplink communication is likely to be slower than the downlink communication speed [124]. To reflect this difference, as demonstrated in [186], the local MSI to be uploaded from each device to the helper can be FD that minimizes the payload size. By contrast, the global MSI to be downloaded to the devices can be the entire model parameters, as used in FSVRG, which may lead to the largest payload size that contains the largest knowledge. Since this global model parameter MSI is not consistent with the uploaded model output MSI, one needs to reconstruct a global model from the model outputs, which can be done via the KD operations with a test data set as exemplified in FD. Consuming the helper's extra computation resource and time can be justified so long as the computation cost is cheaper than the communication cost.

4) Device-to-Device MSI Exchange:

a) Distributed parallel SGD: Exchanging only with neighboring devices can reduce the communication overhead. Namely, with the d-d split, the local MSI of distributed parallel SGD (DSGD) is each device's model weights [105], [187], and its weight update is represented as

$$w_{k+1}^{(i)} = \tilde{w}_k - \eta g(w_k^{(i)}) \tag{28}$$

where the global MSI $\tilde{w}_k$ is the average weight among the communicating devices, i.e., $\tilde{w}_k = \sum_{j=1}^{M} a_{ji} w_k^{(j)} / \sum_{j=1}^{M} w_{ji}$. A predefined mixing matrix determines with whom to exchange the local MSIs, which has the element $w_{ji} = 1$ if the jth device communicates with the ith device, and otherwise, we obtain $w_{ji} = 0$. When the device indices follow their physical locations, neighboring communication topology is characterized by a diagonally clustered weight matrix.

b) Group ADMM: Without the aid of any central entity, group alternating direction method of multiplier (GADMM) exchanges model weights with neighboring devices and achieves fast training convergence with much less communication rounds [106], [183]. The key idea is to apply the ADMM algorithm after grouping devices into the head and tail devices, such that each device in the head group $\mathcal{N}_h$ is connected to two neighboring devices in the tail group $\mathcal{N}_h$. In GADMM, the weights of devices belonging to the same group are updated in parallel, but the weights of devices belonging to different groups are updated in an alternating fashion. When odd and even superscripts denote head and tail devices respectively, for a loss function $\ell(\cdot)$, GADMM updates primal variables (i.e., weights) and dual variables (λ_{n-1} and λ_n) as follows.

1) Each head device updates its primal variables as

$$\begin{aligned} w_{k+1}^{(i\in\mathcal{N}_h)} &= \underset{w_k^{(i)}}{\operatorname{argmin}}\, \lambda_k^{(i-1)}\big(w_k^{(i-1)} - w_k^{(i)}\big) + \lambda_k^{(i)}\big(w_k^{(i)} - w_k^{(i+1)}\big) \\ &\quad + \frac{\rho}{2}\big(\|w_k^{(i-1)} - w_k^{(i)}\|_2^2 + \|w_k^{(i)} - w_k^{(i+1)}\|_2^2\big) \\ &\quad + \ell\big(w_k^{(i)}\big). \end{aligned} \tag{29}$$

These updates are sent to two tail neighbors.

2) Next, each tail device updates its primal variables as

$$\begin{aligned} w_{k+1}^{(i\in\mathcal{N}_t)} &= \underset{w_k^{(i)}}{\operatorname{argmin}}\, \lambda_k^{(i-1)}\big(w_{k+1}^{(i-1)} - w_{k+1}^{(i+1)}\big) + \lambda_k^{(i)}\big(w_k^{(i)} - w_{k+1}^{(i+1)}\big) \\ &\quad + \frac{\rho}{2}\big(\|w_{k+1}^{(i-1)} - w_k^{(i)}\|_2^2 + \|w_k^{(i)} - w_{k+1}^{(i+1)}\|_2^2\big) \\ &\quad + \ell\big(w_k^{(i)}\big). \end{aligned} \tag{30}$$

These updates are sent to two head neighbors.

3) Finally, every device updates its dual variables as

$$\lambda_{k+1}^{(i)} = \lambda_k^{(i)} + \rho\big(w_{k+1}^{(i)} - w_{k+1}^{(i+1)}\big). \tag{31}$$

Consequently, each device communicates with only two neighbors to update its own weights, while only half of devices broadcast their weights per communication round.

c) Federated reinforcement learning: The mixing matrix-based MSI exchange in DSGD is applicable to the MSI exchange in MARL that follows the d-d split. Then, exploiting FL further improves the communication efficiency, leading to federated reinforcement learning (FRL). To illustrate, consider the policy gradient method in MARL,

Table 1 Comparison of MSI Exchanges in Centralized ML and Edge ML. Compared to CSGD as a Baseline Scheme, the Advantages of Each Method Appear Boldfaced, Whereas the Disadvantages Are Italicized

	Centralized ML		**Edge ML**							MARL via **Edge ML**	
	CSGD [101]–[104]	ESGD [109]	**FAvg** [28]	**FSVRG** [30]	**CD** [181]	**FD** [180]	**FJD**	**DSGD** [105]	**GADMM** [183]	**FRL** [184]	**FRD**
Split	m-d	m-d	h-d	h-d	h-d	h-d	h-d	d-d	d-d	d-d	d-d
Tx. MSI	grad.	weight	grad. (**periodic**)	grad., *weight* (periodic)	weight	local avg. **logit/label**	local avg. **Jacob./label**	weight (**neighbor**)	weight (**neighbor**)	grad.	local Q/*state*
Rx. MSI	sample, avg. grad.	sample, avg. weight	avg. grad. (**periodic**)	avg. grad., *weight* (periodic)	avg. weight (periodic)	global avg. **logit/label**	global avg. **Jacob./label**	weight (**neighbor**)	weight (**neighbor**)	avg. grad. (periodic)	global avg. Q/*state*
Comp.	-	-	-	-	*inference*	-	*Jacob.*	-	*optimization*	-	-
Mem.	-	*previous weight*	-	*previous weight*	-	**logit/label, ↓↓**	**Jacob./label, ↓**	-	-	-	Q/*state*

where each agent's policy is trained using SGD with an NN [188]. Following either FAvg or FSVRG, the agents identified in a mixing matrix collaboratively train an ensemble policy by periodically exchanging the NN's model parameters. A similar procedure is applicable to deep Q-learning in MARL, where each agent approximates the Q values using an NN, as demonstrated in [184].

d) Federated reinforcement distillation: FD is applicable to the MSI exchange in MARL with the d-d split, leading to FRD. For Q-learning in MARL, the agents identified in a mixing matrix collectively predict a set of Q values and their associated states. The MSI exchange and weight update rule follow FD, by replacing its normalized logits and labels with the Q values and the states, respectively. Similarly, a policy-based method in MARL can be improved in combination with FD. For actor-critic RL, FD is applicable to either one of the actor (policy) NN or the critic (value) NN or to both NNs [189]. However, with large input state and/or output dimensions, these approaches may incur huge communication and memory costs. To resolve this issue, states and/or actions can be compressed based on their correlations. To illustrate its effectiveness, consider the Atari gaming environment [190] whose output action dimension is only {up, down, left, right}, whereas the input state dimension is the entire pixels per frame. Since neighboring pixels are highly correlated, one can reduce the input dimension by grouping multiple neighboring raw states as a single proxy state that is mapped into the average action of the raw states.

All the aforementioned MSI exchange methods are summarized in Table 1. On top of these methods, one can further enhance communication efficiency, by additionally quantizing gradients [191], removing insignificant gradients, i.e., sparsification [192], opportunistic uploading based on gradient magnitudes [193], and adaptively adjusting the communication intervals [31], [194]. For more details on state dimensionality reduction and advanced FL and FD frameworks in both supervised learning and RL, the readers are encouraged to check [186], [195], and [196].

MSI Exchange Related Theories:

(a) *Optimal Regularization via SGLD:* Distillation-based MSI exchange methods rely on inserting a noise proportionally to the knowledge gap between the teacher and student NNs. The noise amount is adjustable via the temperature parameter T when normalizing the teacher's logit values, and can further be optimized via SGLD. For a fixed learning rate η, a recent work [197] shows that the noise amount maximizing the test accuracy in centralized ML is characterized by the optimal noise scale $g = \eta(n/B - 1)$ of the SGLD's noise ε_t (see Section IV-A), for n training samples and batch size B. The definition of g comes from the autocorrelation of ε_t, given as $\mathsf{E}[\varepsilon_{t+\tau}\varepsilon_t] = gF(w)\delta_\tau$, where $F(w)$ is a matrix describing the covariance of gradients, and δ_τ is the Dirac delta function of τ.

(b) *Wasserstein Distillation Loss:* Cross entropy is widely utilized as the distillation regularizer, which is decomposed into entropy and KL divergence terms. Due to the KL divergence, the distillation regularizer may diverge, particularly when the teacher and students' logits are too peaky to have an overlapping support. A quick fix is to increase the teacher's temperature T for smoothing its logits, at the cost of compromising the accuracy of measuring the knowledge gap (see. Fig. 11). Instead, Wasserstein distance (see Section IV-A) can be a suitable regularizing function for such cases.

C. Computation-Aware Latency Reduction

Compressing model parameters during training operations is effective in the latency reduction, as demonstrated by high-accuracy low-precision (HALP) training that adjusts the arithmetic precision based on the training dynamics [198]. Processing the training operations together with other incumbent applications is another viable solution, in which their operation scheduling is optimized as done in MEC [199] and in the context of exploration–exploitation tradeoff [200].

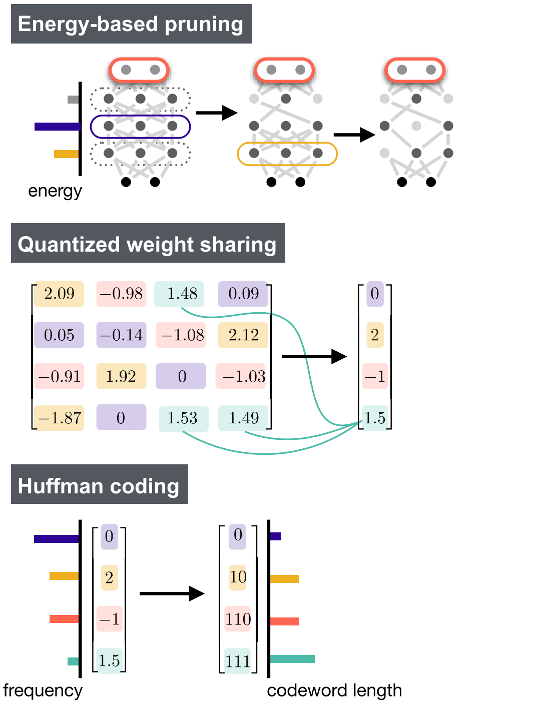
Furthermore, compressing the NN model after completing the training process reduces the inference latency. Such compression can be codesigned with hardware and computational characteristics, such as the energy consumption, compression ratio, and the model parameters' frequency of use, as studied in energy-based pruning [201], Viterbi-based compression [202], and deep compression [113], as detailed next.

1) Adaptive-Precision Training: In mixed-precision training [114], a low-precision representation consists of an exponent δ and mantissa b, and the tuple (δ, b) can express a numerical value within the range $\{-\delta 2^{b-1}, \ldots, -\delta, 0, \delta, \ldots, \delta(2^{b-1} - 1)\}$ by only using $(\delta + b + 1)$ bits, where the last 1 bit is allocated for the sign. Here, the lower precision (i.e., the more compression), the higher quantization noise that increases gradient variance during the training process. To optimize the compression–distortion tradeoff, HALP training is a viable solution [198]. This first utilizes SVRG [177] for reducing the gradient variance. Besides, HALP applies a bit-centering technique that dynamically recenters and rescales the low-precision numbers by setting $\delta = g(\hat{w}_k)/[\mu(2^{b-1} - 1)]$ for a μ-strongly convex loss function where its gradient $g(\hat{w}_k)$ is defined in Section V-B1. This lowers the quantization noise asymptotically as the training converges, thereby achieving the accuracy of a full-precision SVRG. Furthermore, while slightly compromising accuracy, one can represents the gradients only using their ternary directions $\{-1, 0, 1\}$ [203], minimizing the memory usage.

2) Application-Training Coprocessing: In edge ML, a device is likely to perform both the NN training and its end application processing simultaneously. Let us recall the real-time AR/VR application example in Section I, where each headset device predicts the future gaze direction, thereby prerendering future visual frames. In this case, the NN training and the rendering processes are simultaneously performed at the device, and the device's computing energy allocation needs to be optimized under the widely known exploration–exploitation tradeoff [200]. Furthermore, with the helper–device split, a part of the demanding rendering processes can be offloaded from the device to the helper that also participates in the NN training process. At the helper side, its computation energy has to be optimized, as investigated in the context of multiuse MEC [199].

3) Hardware-Efficient Compression: After a training process, compressing the model reduces the memory usage. In this respect, one can prune the model via DropOut and/or DropConnect based on, e.g., Fisher information of each node [117]. Alternatively, one can optimize the pruning process based on energy consumption [201], as shown in Fig. 16. In energy-based pruning, it first estimates the energy consumption per layer and then prune the weights within each layer in order of energy.

After these pruning processes, the resulting weight parameters are expressed as a sparse weight matrix where there exist only a few nonzero values in a large-sized matrix, which can be compressed using, e.g., the compressed sparse row (CSR) format. The compression rate depends on the pruning process, and therefore, a compression-rate optimal pruning is needed. Exploiting the Viterbi algorithm is useful for this purpose, guaranteeing a constant maximum compression rate [202]. On the other hand, in deep compression [113], the weights after pruning are quantized and clustered, yielding a set of shared weights as shown in Fig. 16. Afterward, the shared weights are further compressed using Huffman coding that allocates more bits, i.e., longer codeword length, to the shared weights that appear more frequently.



$$\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ -1.87 & 0 & 1.53 & 1.49 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 2 \\ -1 \\ 1.5 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 2 \\ -1 \\ 1.5 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 10 \\ 110 \\ 111 \end{bmatrix}$$

Fig. 16. ***Illustration of hardware-efficient model compression via energy-based pruning, quantized weight sharing, and Huffman coding.***

D. Scalability Enhancement

Modern deep NN architectures often have too large depths to be stored at mobile devices. This calls for splitting an NN into segments distributed over multiple devices, as shown in Fig. 9(d). Furthermore, the range of federation in edge ML is constrained by the battery levels and privacy requirements of mobile devices, and the effectiveness of federation is delimited by its non-IID training data set. These challenges and their suitable solutions are described as follows.

1) Stacked Model Split: In the model split, the data may flow back and forth between the split model segments. The resulting dependence among the segments stored in multiple devices obstructs the parallelism of local training. To minimize such dependence, one can start from the original model comprising multiple stacks of components that can be easily parallelized. A suitable example can be a discriminator distributed GAN [204] whose discriminators can be distributed over multiple devices. In the opposite way, multiple generators can be distributed over the devices that share a single discriminator [205]. Both

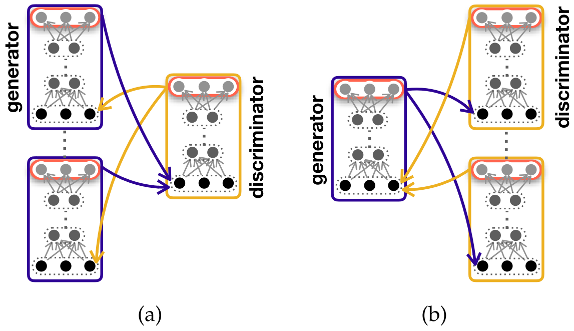


Fig. 17. *Illustration of a distributed GAN with (a) multiple generators and (b) multiple discriminators.*

cases are shown in Fig. 17. Another example is an DBN that comprises multiple stacked RBMs distributed over several devices. More general model split-based edge ML frameworks are discussed in [206] and [207], in the context of health applications with private patient data.

2) ODE-Based Training: Traditional ML cannot dynamically change the model size during the training process, so it is difficult to flexibly adjust the energy consumption. This is critical to mobile devices whose battery level is limited and fluctuates over time, which hinders adopting edge ML into more devices. One promising solution for this bottleneck is to train an NN by solving an ordinary differential equation (ODE) [208]. In this framework, adjusting the model size is recast as changing the number of evaluations that can be easily feasible in practice. Furthermore, compared to SGD rooted in convex optimization methods, ODE-based training is able to directly solve a nonconvex optimization, avoiding local minima issues, as addressed in Section III-A2. To this end, one can utilize an algorithmic approach by exploiting the feedforward dynamics from the lth layer to the next layer with a generic loss function $\mathcal{L}$

$$x_{l+1} = x_l + \mathcal{L}(x_l, w_l), \quad l \in \{0, \ldots, L-1\}. \tag{32}$$

If the number L of layers is sufficiently large, (32) is approximated as the following ODE:

$$dx(l)/dl = \mathcal{L}(x(l), w(l)), \quad l \in [0, L]. \tag{33}$$

This ODE can be numerically solved via Euler's approximation method with the approximation accuracy proportional to the number of evaluations [168].

ODE-Based Training Related Theories:

(a) *MF Controlled Training:* One can analytically solve (33) with the empirical loss function with a regularizer R.

$$\mathcal{L}(w) = \frac{1}{n}\sum_{i=1}^{n}\left[\Phi(x_L^i, y_o^i) + \int_0^L R(x_l^i, w_l)dt\right] \tag{34}$$

Here, the weight parameters w_l is shared by a number n of the training data samples. So long as n is sufficiently large, according to mean-field control theory, the minimized loss, i.e., value function, satisfies Pontryagin's maximum principle (PMP). The optimal solution thereby guarantees a necessary condition that is recast as maximizing the Hamiltonian:

$$H(x_o, w) = -\nabla H(x_o, w)\mathcal{L}(x, w) - \mathcal{R}(x, w). \tag{35}$$

Particularly when $H(x_o, w)$ is strongly concave, by setting $T \to 0$, this solution guarantees the global optimum.

3) Private Data Augmentation: Due to the user-generated data samples, training data set across devices can be non-IID in edge ML, which severely degrades the benefit of distributed training. A simple example is the situation when all devices have identical data samples, i.e., fully correlated. In this case, the global and local MSI become identical, negating the diversity gain from distributed training. At the opposite extreme, if all the data sets are entirely not correlated, then reflecting the global MSI in the local weight update is no more than inserting a randomly noisy regularizer. Data augmentation can render such a non-IID data set amenable to distributed learning. One possible implementation is partially exchanging the other devices' data samples. In fact, an experimental study [179] has shown that FL under a non-IID data set achieves only 50% inference accuracy compared to the case under an IID data set, which can be restored by up to 20% via randomly exchanging only 5% of the devices' local training data samples. Another way is locally augmenting data samples. This is viable, for instance, by a generative model that is capable of generating all samples, which can rectify the non-IID data set toward achieving an IID data set across devices.

Both approaches require access to data samples owned by other devices, thus necessitating privacy guarantee. Data samples can be exchanged while preserving privacy via DP by partially inserting noise and/or redundant data samples. Local data oversampling needs to exchange the local data sample distribution to collectively construct the entire data set distribution that is to be compared with the local data sample distributions. Such distribution information, including any excess or shortage of data samples per label, e.g., per medical checkup item, may easily reveal private sensitive information, e.g., diagnosis result. A GAN-based solution [180] for this case is elaborated in Section VI-D.

VI. CASE STUDIES

From the standpoint of CML, this section aims at demonstrating the effectiveness of the proposed theoretical and technical solutions in edge ML. Several use cases that follow MLC are also introduced at the end while addressing their connections to MLC.

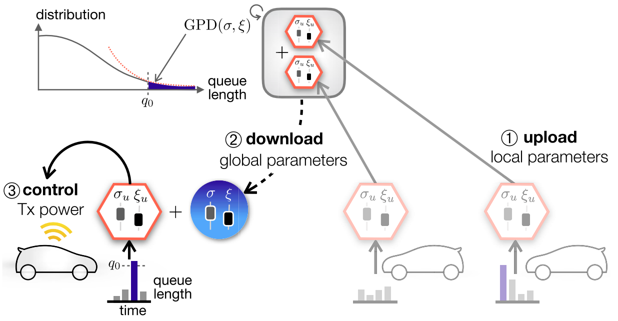
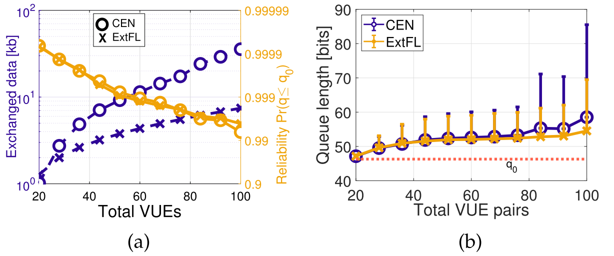


Fig. 18. ***Operational structure of EVT parametric FL (ExtFL).***



(a)

(b)

Fig. 19. ***Comparison between CEN and ExtFL. (a) Amount of data exchanged between RSU and VUEs (left) and the achieved reliability (right). (b) The mean and the variance of the worst-case VUE queuelengths.***

A. Federated Learning With EVT for Vehicular URLLC

EVT-based FL enables URLLC in vehicular communication networks as discussed in our preliminary study [209]. EVT provides an analytical parametric model to study the tail distribution of queue lengths at vehicular transmitters over the whole network. By combining the parametric model from EVT with FL, referred to as ExtFL, the individual vehicles learn the tail distribution of queue lengths over the network without a need of exchanging the local queue length samples. Therein, the key advantage of FL is the reduction of communication payload during the model training compared to a centralized training model relying on exchanging the training samples. In this regard, the impact of communication latency for training on the vehicular-to-vehicular (V2V) communication is reduced.

The objective is to minimize the network-wide power consumption of a set $\mathcal{U}$ of vehicular users (VUEs) while ensuring low queuing latencies with high reliability. Yet, there exist worst case VUEs who are experiencing high latencies with a low probability. In this regard, extreme events pertaining to vehicles' queue lengths exceeding a predefined threshold with nonnegligible probability are considered to capture the performance losses of worst case VUEs. Using the principles of EVT, the tail distribution of the queue lengths exceeding a predefined threshold is characterized by a GPD $G^{\boldsymbol{d}}(\cdot)$ with two parameters $\boldsymbol{d} = [\sigma,\xi]$ scale and shape, respectively. The knowledge of the tail distribution over the network is utilized to optimize the transmit power of each VUE to reduce the worst case queuing delays.

To estimate the queue tail distribution using the queue length samples $\{\mathcal{Q}_u\}_{u\in\mathcal{U}}$ observed at each VUE u, using the concepts in maximum likelihood estimation (MLE), a cost function is defined as follows:

$$f^{\boldsymbol{d}}(\mathcal{Q}) = \frac{1}{\sum_u |\mathcal{Q}_u|}\sum_{u\in\mathcal{U}}\sum_{Q\in\mathcal{Q}_u}\log G^{\boldsymbol{d}}(Q) = \sum_{u\in\mathcal{U}}\kappa_u f^{\boldsymbol{d}}(\mathcal{Q}_u) \tag{36}$$

where $\kappa_u = (|\mathcal{Q}_u|/\sum_u |\mathcal{Q}_u|)$. The operation of ExtFL is visualized in Fig. 18 and summarized as follows.

1) VUE u uses $f_u^{\boldsymbol{d}} = f^{\boldsymbol{d}}(\mathcal{Q}_u)$ to evaluate $\boldsymbol{d}_u$ and $\nabla_{\boldsymbol{d}} f_u^{\boldsymbol{d}_u}$ locally, where $\boldsymbol{d}_u$ is the local estimate of $\boldsymbol{d}$ at VUE u. Then, the local learning model $(\nabla_{\boldsymbol{d}} f_u^{\boldsymbol{d}_u}, \boldsymbol{d}_u, |\mathcal{Q}_u|)$ is uploaded to the road-side unit (RSU).
2) RSU does the model averaging and shares the global model $(\nabla_{\boldsymbol{d}} f^{\boldsymbol{d}}, \boldsymbol{d}, \sum_u |\mathcal{Q}_u|)$ with the VUEs.
3) VUEs use the global parameters to model the tail distribution of queue lengths and utilize it to control their transmit powers.

Fig. 19(a) compares the amount of data exchanged and the achieved V2V communication reliability of ExtFL with a centralized tail distribution estimation model, denoted as CEN. Fig. 19(a) shows that VUEs in ExtFL achieve slightly lower reliability compared to the ones in the CEN approach for $U < 72$ while outperforming CEN when $U > 72$. Note that the CEN method requires all VUEs to upload all their queue length samples to the RSU and to receive the estimated GPD parameters. In contrast, in ExtFL, VUEs upload their locally estimated learning models $(\nabla_{\boldsymbol{d}} f_u^{\boldsymbol{d}_u}, \boldsymbol{d}_u, |\mathcal{Q}_u|)$ and receive the global estimation of the model. As a result, ExtFL yields equivalent or better end-user reliability compared to CEN for denser networks while reducing the amount of data exchange among VUEs and RSU by 79% when $U = 100$.

The worst case VUE queue lengths, i.e., queue lengths exceeding q_0, are compared in Fig. 19(b). Here, the mean and variance of the tail distribution for CEN and ExtFL are plotted for different numbers of VUEs. The mean indicates the average queuing latency of the worst case VUEs, while the variance highlights the uncertainty of the latency. As the number of VUEs increases, it can be noted that both the mean and the variance in ExtFL are lower than the ones in CEN. The reason for the above-mentioned improvement is the reduced training latency in ExtFL over CEN.

B. Federated Learning With Wasserstein Distances

The Wasserstein distance can precisely measure the similarity between two distributions even when they have nonoverlapping support. To show its effectiveness, consider the same application in Section VI-A while replacing its MLE-based tail distribution estimation with the Wasserstein distance-based estimation, i.e., Wasserstein-based FL.

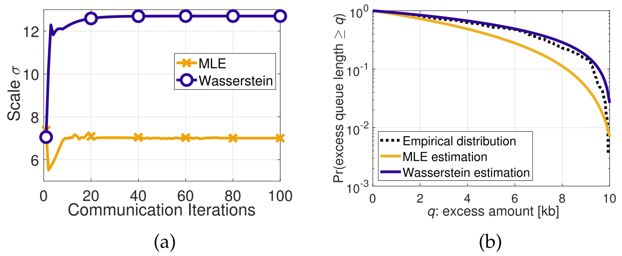


Fig. 20. ***Comparison of GPD parameter estimation and tail distribution estimation between MLE and Wasserstein distance. (a) Scale parameter. (b) Empirical and estimated tail distributions.***

Accordingly, we redefine the cost function $f^{\boldsymbol d}(Q)$ in (36) with the Wasserstein distance given in (16). In this regard, the local and global models defined for FL require to compute the gradient as follows [210]:

$$\nabla_{\boldsymbol d} f_u^{\boldsymbol{d}_u}(Q) = -\tfrac{1}{\kappa_u}\alpha^\star(Q)\nabla_{\boldsymbol d} F_u^{\boldsymbol{d}_u}(Q) \tag{37}$$

where $\alpha^\star(Q)$ is the dual function of the corresponding primal problem defined in (16). Here, the function $F_u^{\boldsymbol{d}_u}$ satisfies $G^{\boldsymbol d}(x') = e^{-F_u^{\boldsymbol{d}_u}(x')}/Z$ with the partition sum $Z = \sum_{x'} e^{-F_u^{\boldsymbol{d}_u}(x')}$. By utilizing the knowledge of the parametric representation of $G^{\boldsymbol d}(x')$, the function $F_u^{\boldsymbol{d}_u}$ can be derived, while $\alpha^\star(Q)$ is calculated using the Sinkhorn algorithm [154].

Fig. 20 corroborates the advantage of Wasserstein-based FL compared with MLE-based FL in Section VI-A. Compared with MLE-based FL, Fig. 20(a) shows that Wasserstein-based FL converges as fast as MLE-based FL, but with a different converging point compared with MLE-based FL. Fig. 20(b) validates that the converging point of Wasserstein-based FL is closer to the optimum, thereby more accurately estimating the tail distribution of the queue lengths exceeding a target threshold. The higher accuracy of Wasserstein-based FL results from the fact that the Wasserstein distance counts the differences between the empirical and parametric distributions over the entire supports, whereas the KL divergence in MLE ignores the differences over only the points at which the parametric distributional values are sufficiently large [78].

Finally, Fig. 21 shows the impact of local computing iterations on the training convergence. In Fig. 21(a) and (b), reducing local SVRGD iterations yields faster training convergence because of exchanging the model parameters more frequently. However, as it exchanges the parameters of less trained models, the converging points fall farther away, lowering the tail distribution estimation accuracy. This relationship highlights the importance of optimizing local computing and global communication iterations.

C. Federated Learning With Blockchain

The reliability and scalability of FL can further be improved by adopting blockchain [211], [212], as exemplified by block-chained FL (BlockFL) in our preliminary study [172]. BlockFL provides incentives to devices that own a larger number of local training samples and consume more computing power, which promotes federation with more devices. In addition, local training results in BlockFL are mutually validated, thereby extending the range of federation to untrustworthy devices in a public network. All these operations as well as local MSI exchanges are fully decentralized, which is more robust against malfunctions and attacks compared with the original FL [28], [30] that relies on a single helper entity.

As shown in Fig. 22, the logical structure of BlockFL consists of devices and miners. Miners can physically be either randomly selected devices or separate nodes such as a conventional blockchain network [211]. The operation of BlockFL is summarized as follows.

1) Each device computes and uploads the local MSI to its associated miner in the blockchain network while, in return, receiving the data reward proportional to the number of its data samples from the miner.
2) Miners exchange and verify all the MSIs and then run the proof of work (PoW) [211].
3) Once a miner completes the PoW, it generates a block where the verified local MSIs are recorded and receives the mining reward from the blockchain network. The generated block is propagated and added to every miner's ledger.
4) Finally, the ledgers are downloaded to the miners' associated devices. Each device locally computes the global MSI from the freshest block, which becomes an input of the next local model update.

At step 3, every miner is enforced to stop its PoW process once it receives a propagated block that should be the earliest generated block, thereby synchronizing the distributed ledgers. However, if a miner generates a block during the earliest generated block's propagation delay, this miner unknowingly adds its own generated block to the ledger that becomes different from the other legitimate ledgers. This forking event incurs extra delays for rolling the unsynchronized ledger back. When the block generation rate λ of each miner is centrally controlled by adjusting the PoW difficulty, the optimal block generation rate λ^* is thus obtained by balancing between the forking

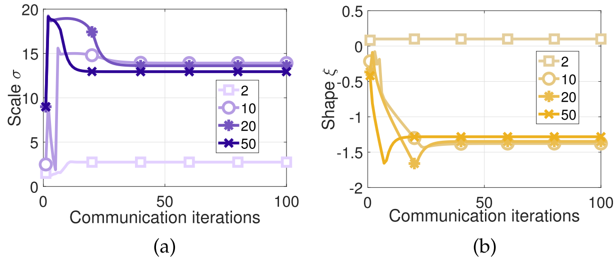


Fig. 21. ***Convergence speed for different local SVRGD iterations. (a) Scale parameter. (b) Shape parameter.***

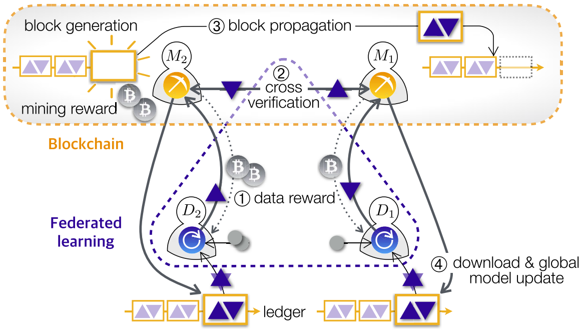


Fig. 22. ***Operational structure of BlockFL.***

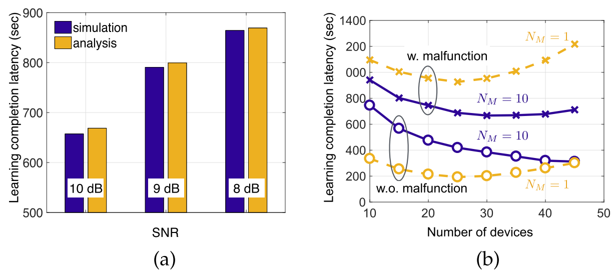
occurrences and the block propagation delays, which is approximated as

$$\lambda^* \approx 2\left(T_{\text{bp}}\left[1+\sqrt{1+4N_M(1+T_{\text{wait}}/T_{\text{bp}})}\right]\right)^{-1} \quad (38)$$

where N_M is the number of devices, T_{bp} is the longest propagation delay of the legitimate block, and T_{wait} is the maximum waiting time before starting the PoW process.

Fig. 23(a) plots the E2E latency of BlockFL with λ^* until the local model parameters converge, which involves all delays incurred by FL and blockchain operations. The simulation parameters follow the 3GPP LTE Cat. M1 specification [213] with $N_M = 10$. As the received signal-to-noise ratio (SNR) decreases from 10 to 8 dB, both uplink/downlink MSI delays and block propagation delays increase, and we therefore observe the increased E2E latency. Compared to the simulated optimum, the E2E latency with (38) shows only up to 1.5% difference.

Fig. 23(b) shows the scalability and robustness of BlockFL. Without any malfunction, a larger N_M value increases the latency due to the increase in their cross verification and block propagation delays. This does not always hold under the miners' malfunctions that are captured by adding a Gaussian noise $\mathcal{N}(-0.1, 0.01)$ to each miner's aggregate MSIs with the probability of 0.5. In BlockFL, global MSI is locally calculated at each device, and each miner's malfunction thus only distorts its associated device's MSI. Such distortion can be restored by federating with other devices that associate with the miners operating normally. For this reason, a larger N_M value may achieve even shorter latency, as observed for $N_M = 10$ with malfunctions.



(a) (b)

Fig. 23. ***Average learning completion latency with the optimum block generation rate λ^*. (a) For different SNRs. (b) With/without malfunction.***

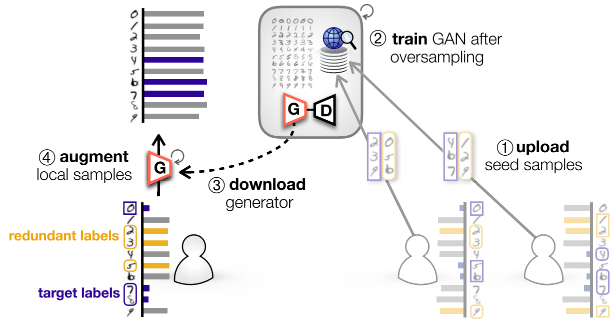


Fig. 24. ***Operational structure of FAug.***

D. Federated Augmentation Rectifying Non-IID Data

The non-IID training data set of on-device ML can be corrected by obtaining the missing local data samples at each device from the other devices [179]. Such a sample exchange may, however, induce significant communication overhead, especially with a large number of devices, and may violate the privacy requirement of data samples. Instead, federated training and exchanging a generative model can rectify the non-IID data set by enabling each device to locally augment the missing data samples while abiding by a target privacy requirement. This method and its scalability are demonstrated by federated augmentation (FAug) in our previous study [180].

The procedure of FAug is shown in Fig. 24, which is described as follows. At first, each device recognizes the labels being lacking in data samples, referred to as target labels, and uploads a few samples of these target labels, referred to as seed samples, to the helper over wireless links. Then, the generative model, a part of a conditional GAN [214], is trained at a helper with high computing power and a fast connection to the Internet. GAN training commonly requires a large number of training samples. With this end, the helper oversamples the uploaded seed samples, e.g., via Google's image search for visual data, and thereby trains the GAN. Finally, downloading the trained GAN's generator empowers each device to replenish the target labels until reaching an IID training data set.

The operation of FAug needs to guarantee privacy of the user-generated data. In fact, each device's data generation bias, i.e., target labels, may easily reveal its privacy-sensitive information, e.g., patients' medical checkup items revealing the diagnosis result. To keep these target labels private from the helper, the device additionally uploads

Table 2 Test Accuracy and Communication Cost of FAug (Single Target Label, No Redundant Label)

Methods	Accuracy w.r.t. the number of devices					Communication cost		
	2	4	6	8	10	logits	model param.	samples
FD + FAug	0.8464	0.8526	0.8498	0.8480	**0.8642**	3,200	1,493,520	5
FD (non-IID)	0.7250	0.7428	0.6951	**0.7891**	0.7524	3,200	-	-
FL + FAug	0.9110	0.8654	0.8974	0.9247	**0.9259**	-	39,882,256	5
FL (non-IID)	0.9032	0.8982	0.8738	**0.9171**	0.9060	-	38,388,736	-

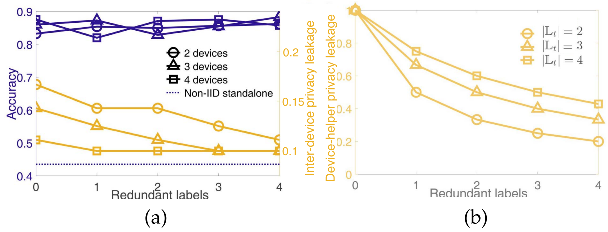


Fig. 25. ***Test accuracy and PL under a non-IID MNIST data set. (a) accuracy and inter-device PL. (b) Device-server PL with respect to the number of redundant labels.***

the redundant data samples from the labels other than the target labels. The privacy leakage (PL) from each device to the helper, denoted as device-helper PL, is thereby reduced at the cost of extra uplink communication overhead. At the ith device, its device-helper PL is measured as $|\mathbb{L}_t^{(i)}|/(|\mathbb{L}_t^{(i)}| + |\mathbb{L}_r^{(i)}|)$, where $|\mathbb{L}_t^{(i)}|$ and $|\mathbb{L}_r^{(i)}|$ denote the numbers of target and redundant labels, respectively.

The target label information of a device can also be leaked to the other devices since they share a collectively trained generator. Indeed, a device can infer the others' target labels by identifying the generable labels of its downloaded generator. This PL is quantified by interdevice PL. Provided that the GAN is always perfectly trained for all target and redundant labels, the interdevice PL of the ith device is defined as $|\mathbb{L}_t^{(i)}|/\bigcup_{j=1}^{M}(|\mathbb{L}_t^{(j)}| + |\mathbb{L}_r^{(j)}|)$. Note that the interdevice PL is minimized when its denominator equals to the maximum value, i.e., the number of the entire labels. This minimum leakage can be achieved so long as the number of devices is sufficiently large, regardless of the sizes of the target and redundant labels.

Table 2 provides the test accuracy and communication cost of FD and FL with or without FAug. For FD, the communication cost is defined as the number of exchanged logits, whereas the cost for FL is given as the number of exchanged model parameters. With FAug, the communication cost comprises the number of uploading data samples and the number of downloading model parameters of the trained generator. We observe that FAug is effective for both FL and FD, improving the test accuracy by 0.8%–2.7% for FL and by 7%–22% for FD. Such a gap between the improvements implies that FD is more vulnerable to the non-IID data set than FL. In FD, even if a device obtains the full teacher's knowledge across all labels, the distillation operation cannot be performed when the device has no local training sample in target labels, which is undesirable.

Overall, we observe that FL achieves the highest test accuracy while consuming significant communication cost due to exchanging a large number of model parameters. In combination with FAug, FD can cope with the non-IID data set and achieve 92%–97% accuracy of FL, In this case, the aggregate communication cost of FD and FAug is up to 25.6× smaller than FL, highlighting the communication efficiency of FD.

Fig. 25(a) shows that increasing the number of devices makes the target label uploaders anonymous, thereby reducing the interdevice PL while preserving the test accuracy. More redundant labels allow the uploaders to hide their target labels, which also decreases the interdevice PL. Likewise, the device-helper PL decreases with the number of redundant labels, as shown in Fig. 25(b). Alternatively, multihop communication from each device to the server is also effective in hiding target label privacy in the crowds of preceding hops' devices. Its impact on FAug is elaborated in [215].

E. Field-of-View Prediction for Multicast VR Streaming

GRUs, a gating mechanism in RNNs, are commonly used for modeling speech signals and musics. Their pattern recognition capability can be utilized for predictions that allow proactive control for latency-sensitive applications. In this aspect, proactive content quality adaptation for multiuser 360° VR video streaming based on the field-of-view (FoV) prediction using GRUs is studied in [216].

The scenario is a VR theater consisting of a network of VR users watching different HD 360° VR videos streamed over a set of distributed small cell BSs (SBSs). SBSs operate in the mmWave band and are multibeam beamforming capable to improve multicast transmission of shared video content to groups of users. For user grouping, as Fig. 26 illustrates, upcoming tiled FoV predictions obtained via a deep NN architecture based on GRUs are utilized. By optimizing video frame admission and user scheduling, the goal is to provide highly reliable broadband service for VR users that deliver HD videos with low latency.

Fig. 27 evaluates the impact of the video quality corresponding to the delivered frame rates. Therein,

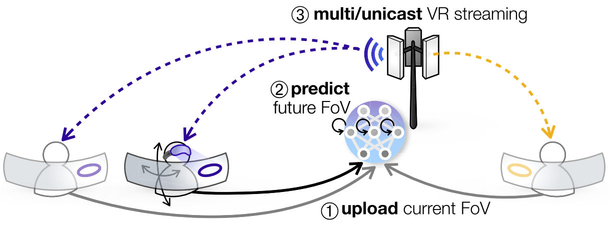


Fig. 26. ***Operational structure of the 360° VR video streaming via mmWave multicast (blue) and unicst (yellow) transmissions.***

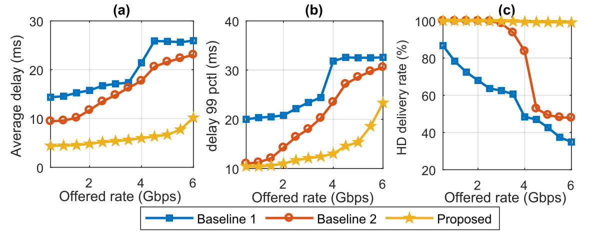


Fig. 27. ***Performance comparison with respect to (a) average delay, (b) 99th-percentile delay, and (c) HD delivery rate.***

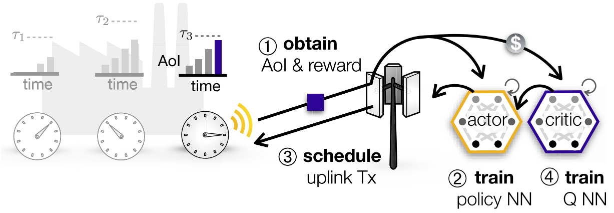


Fig. 28. ***Operational structure of the actor-critic RL for optimizing the AoI.***

the proposed FoV prediction-based scheme is compared with two benchmark methods: Baselines 1 and 2 that schedule video chunk requests in real time over mmWave unicast and multicast transmissions, respectively. Fig. 27(a) and (b) shows that the FoV-based predictions allow the proposed proactive scheduler to decrease both average and 99th percentile delays over the baseline methods. While both the baseline schemes reduce the HD delivery rates for increased offered data rates as shown in Fig. 27(c), the proposed approach maintains about 100% success HD delivery rate by predicting video frames in advance.

As the predictions become accurate, the need of retransmissions reduces and a surplus of wireless resources can be smartly reused to feed back prediction event errors.

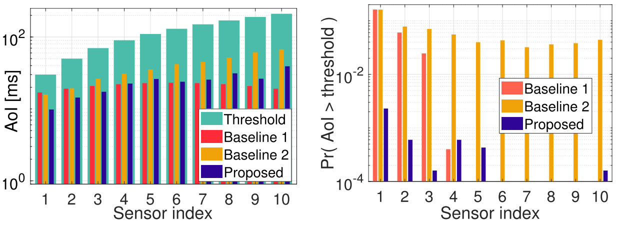
F. Actor-Critic RL for Optimizing Age of Information

Age of information (AoI) is a measure of the freshness of data to characterize the E2E communication latency. While minimizing AoI enables URLLC, the performance highly depends on the availability of the system state knowledge, such as physical resources, channel conditions, packet drops, and sampling. As a remedy, RL can be adopted to explore and learn the system state dynamics and improve the process of decision-making over time. In this regard, minimizing AoI of remote sensors with the aid of an RL-based scheduler is presented in [217].

As shown in Fig. 28, the scenario is focused on a set of sensors in a factory randomly generating data packets and remotely monitored. To ensure high reliability and low latency of the received sensor data at the remote monitor, the controller schedules sensors to report their data. Due to the lack of knowledge of the conditions of the communication links, packet generation at the sensors, and losses during communication, the monitoring unit resorts to an RL-based scheduler. Here, the states observed at the remote monitoring unit are the AoI of each sensor (τ_i), previous data rates, and the time spent to download the last data packet. Based on that, RL scheduler builds a probability distribution over the sensors, which is used for scheduling the sensors. A cost for each action is defined in terms of the average AoI over the sensors and aggregated penalties when sensors' AoI exceed their predefined thresholds. The RL scheduler is trained using an asynchronous advantage actor-critic (A3C) algorithm [218] in an offline manner. In A3C, several copies of the actor agent are trained in parallel (asynchronous) to improve the efficiency using discount functions that indicate gains/losses over average q-values (advantage), while the critic NN estimates the cost function. Here, the trained actor NN is used as the scheduler.

Fig. 29 compares the per-sensor performance of the RL-based scheduler with two baselines: Baseline 1 schedules a sensor with maximum AoI at a given time, while Baseline 2 randomly schedules sensors proportional to the inverse of their AoI thresholds. Here, the system consists of ten sensors indexed by $i = 1, 2, \ldots, 10$. Fig. 29(a) shows that all three methods perform almost the same in terms of average AoI. While all sensors in Baseline 2 exhibit higher average AoI compared to the rest, Baseline 1 and proposed methods display lowest AoI for sensors with loose and tight thresholds, respectively. The probability of AoI exceeding the threshold for each sensor is shown in Fig. 29(b). Therein, it can be noted that the proposed method maintains much lower AoI violation probability for sensors with tight AoI thresholds compared with both baseline methods. For sensors with large thresholds, both the proposed and Baseline 1 exhibit no AoI violations. From Fig. 29(a) and (b), we can conclude that the proposed RL-based scheduler is the most reliable approach for sensors with tight thresholds.

By minimizing the AoI over all sensors, the proposed solution allows the remote monitor to obtain up-to-date



(a) (b)

Fig. 29. ***Per-sensor performance comparison with respect to (a) average AoI and (b) AoI violation probability.***

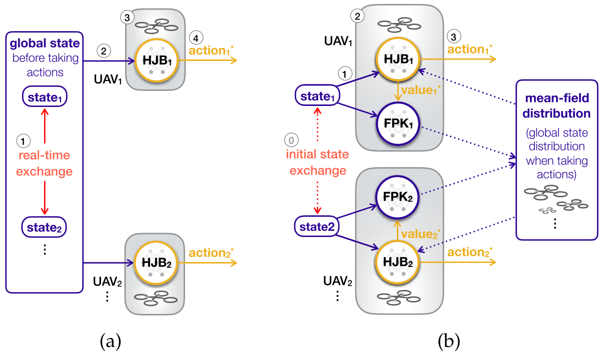


Fig. 30. ***Operational structures of (a) HJB learning control and (b) MFG learning control.***

data. For the applications of monitoring and controlling, these data can be utilized to train control modules and/or to issue real-time commands.

G. MFG Theoretic ML for Massive UAV Control

MFG theoretic control is a key enabler for supporting robust massive autonomous UAV operations against time-varying network connectivity. To illustrate its effectiveness, following [219], we consider N UAVs, each of which locally controls its acceleration so as to minimize its travel time from the same source to a common destination, without inter-UAV collisions. To avoid collisions, each UAV communicates with the other $N-1$ UAVs for collecting their real-time states, i.e., coordinates and velocities. To guarantee the optimality of every control, since each control affects the other UAVs' decisions, the states should be recursively exchanged until all controls converge to the Nash equilibrium. This is likely to be infeasible for a large N value, under the limited communication range of each UAV moving in real time.

This is where MFG theoretic control comes to the rescue, which requires initial state exchanges only once. Afterward, as elaborated in Section IV-B, the control of each UAV is determined by locally solving two partial differential equations, in which the FPK equation ($F = 0$) approximates the state distributions of the entire population following optimal controls, i.e., MF distribution, while the HJB equation ($H = 0$) provides each UAV's optimal control for a given population distribution. However, solving these equations is computationally challenging particularly for high-dimensional states, limiting its adoption commonly within the 1-D state cases [165]–[167].

To overcome this limitation, an NN-based MFG theoretic control algorithm is developed in [219], denoted as MFG leaning control operated by a pair of HJB and FPK NNs, as shown in Fig. 30(b). At its core, the HJB and FPK NNs are trained so as to minimize $|H|$ and $|F|$, thereby approximately solving the HJB and FPK equations, respectively. On the one hand, focusing on the relation between HJB and FPK NNs, this ML architecture is similar to actor-critic RL in which the actor NN (FPK NN or policy NN) yields per-state action distributions, and the critic NN (HJB NN or value NN) evaluates the optimality of controls. On the other hand, focusing on the process of taking actions, MFG learning control resembles DQN whose actions are taken by the Q NN (value NN). Integrating these two RL architectures, MFG learning control addresses the interplay between the population's policy and a single agent's action evaluation, as opposed to traditional RL considering the same agent's policy and action evaluation.

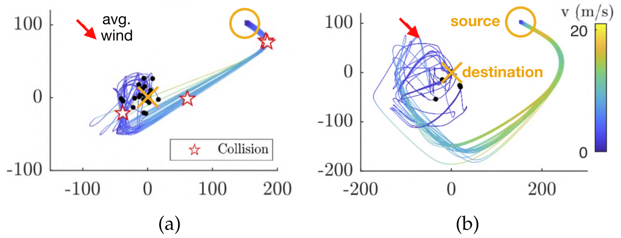


Fig. 31. ***Trajectories of 25 UAVs under (a) HJB learning control and (b) MFG learning control.***

Consequently, the source-to-destination travel trajectory in Fig. 31(b) corroborates that MFG learning control achieves both the goals of collision avoidance and fast travel. This is compared with a baseline HJB learning control in Fig. 31(a), in which UAVs are controlled based not on FPK NNs but on real-time state exchanges. Due to the limited transmission power, the inter-UAV state exchanges are restricted within the communication range, i.e., 1 m, which is not always sufficient to cover $N-1$ mobile UAVs. This incurs time-varying UAV network connectivity and makes HJB learning control fail to avoid collision. By contrast, MFG learning control is effective in collision avoidance, by forming a flock of UAVs. In return, the travel distances under MFG learning control become longer. Nevertheless, because of flocking behaviors, MFG learning control achieves faster speed during long flights. This compensates the longer travel distances, thereby achieving the travel time almost as fast as the HJB learning control ignoring collision avoidance.

VII. CONCLUSION

Embracing recent advances in hardware and communication technologies, edge ML empowers devices at the network edge by imbuing them with the state-of-the-art ML techniques. This poses a slew of new research questions centered on E2E latency, reliability, and scalability under hardware and privacy constraints. As a first step toward spearheading the edge ML vision and moving beyond centralized and cloud-based ML, this article has explored its key building blocks and theoretical principles warranting a clean-slate design in terms of NN architectures, training and inference operations, and communication. The overarching goal of this article is to foster more fundamental research in edge ML and bridge connections among several communities and mathematical disciplines. ■

REFERENCES

[1] *Study on New Radio Access Technology: Radio Access Architecture and Interfaces, Release-14*, document 3GPP TR 38.801 Mar. 2017.

[2] *IMT Vision—Framework and Overall Objectives of the Future Development of IMT for 2020 and Beyond*, document Recommendation M.5/BL/22, 2015.

[3] *Recommendations for NGMN KPIs and Requirements for 5G*, NGMN Alliance, Frankfurt, Germany, Jun. 2016

[4] P. Popovski *et al.*, "Wireless access for ultra-reliable low-latency communication: Principles and building blocks," *IEEE Netw.*, vol. 32, no. 2, pp. 16–23, Mar./Apr. 2018.

[5] M. Bennis, M. Debbah, and V. Poor, "Ultrareliable and low-latency wireless communication: Tail, risk, and scale," *Proc. IEEE*, vol. 106, no. 10, pp. 1834–1853, Oct. 2018.

[6] U. Aiman and V. P. Vishwakarma, "Face recognition using modified deep learning neural network," in *Proc. 8th ICCCNT*, New Delhi, India, Jul. 2017, pp. 1–5.

[7] L. Wan, N. Liu, H. Huo, and T. Fang, "Face recognition with convolutional neural networks and subspace learning," in *Proc. 2nd Int. Conf. Image, Vis. Comput.*, Chengdu, China, Jun. 2017, pp. 228–233.

[8] N. G. Maity and S. Das, "Machine learning for improved diagnosis and prognosis in healthcare," in *Proc. IEEE Aerosp. Conf.*, Big Sky, MT, USA, Mar. 2017, pp. 1–9.

[9] J. Ker, L. Wang, J. Rao, and T. Lim, "Deep learning applications in medical image analysis," *IEEE Access*, vol. 6, pp. 9375–9389, 2018.

[10] S. Lakhanpal, A. Gupta, and R. Agrawal, "Discover trending domains using fusion of supervised machine learning with natural language processing," in *Proc. 18th Fusion*, Washington, DC, USA, Jul. 2015, pp. 893–900.

[11] P. Yang and Y. Chen, "A survey on sentiment analysis by using machine learning methods," in *Proc. IEEE ITNEC*, Chengdu, China, Dec. 2017, pp. 117–121.

[12] S.-C. Lin *et al.*, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proc. 23rd ACM ASPLOS*, Williamsburg, VA, USA, Mar. 2018, pp. 751–766.

[13] M. K. Abdel-Aziz, C.-F. Liu, S. Samarakoon, M. Bennis, and W. Saad, "Ultra-reliable low-latency vehicular networks: Taming the age of information tail," in *Proc. GLOBECOM*, Abu Dhabi, UAE, Dec. 2018, pp. 1–7.

[14] J. Park and M. Bennis, "URLLC-eMBB slicing to support VR multimodal perceptions over wireless cellular systems," May 2018, *arXiv:1805.00142*. [Online]. Available: https://arxiv.org/abs/1805.00142

[15] ABI Research and Qualcomm, "Augmented and virtual reality: The first wave of 5G killer apps," ABI Res., New York, NY, USA, White Paper, Feb. 2017.

[16] T. Kagawa *et al.*, "A study on latency-guaranteed multi-hop wireless communication system for control of robots and drones," in *Proc. 20th WPMC*, Yogyakarta, Indonesia, Dec. 2017, pp. 417–421.

[17] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Unmanned aerial vehicle with underlaid device-to-device communications: Performance and tradeoffs," *IEEE Trans. Wireless Commun.*, vol. 15, no. 6, pp. 3949–3963, Jun. 2016.

[18] A. Fotouhi *et al.*, "Survey on UAV cellular communications: Practical aspects, standardization advancements, regulation, and security challenges," Sep. 2018, *arXiv:1809.01752*. [Online]. Available: https://arxiv.org/abs/1809.01752

[19] S. Dörner, S. Cammerer, J. Hoydis, and S. ten Brink, "Deep learning-based communication over the air," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 132–143, Feb. 2018.

[20] E. Balevi and J. G. Andrews, "One-bit OFDM receivers via deep learning," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4326–4336, Jun. 2019.

[21] N. Farsad and A. Goldsmith, "Neural network detection of data sequences in communication systems," *IEEE Trans. Signal Process.*, vol. 66, no. 21, pp. 5663–5678, Nov. 2018.

[22] H. Ye, G. Y. Li, B.-H. F. Juang, and K. Sivanesan, "Channel agnostic end-to-end learning based communication systems with conditional GAN," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Abu Dhabi, UAE, Dec. 2018, pp. 1–5.

[23] M. Rebato, J. Park, P. Popovski, E. D. Carvalho, and M. Zorzi, "Stochastic geometric coverage analysis in mmWave cellular networks with realistic channel and antenna radiation models," *IEEE Trans. Commun.*, vol. 67, no. 5, pp. 3736–3752, May 2019.

[24] X. Li, A. Alkhateeb, and C. Tepedelenlioglu, "Generative adversarial estimation of channel covariance in vehicular millimeter wave systems," in *Proc. ACSSC*, Pacific Grove, CA, USA, Feb. 2019, pp. 1572–1576.

[25] Y. Wang, M. Narasimha, and R. W. Heath, Jr., "MmWave beam prediction with situational awareness: A machine learning approach," *IEEE Access*, vol. 7, pp. 87479–87493, Jun. 2019.

[26] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Commun. Surveys Tuts.*, to be published.

[27] O. Simeone, *A Brief Introduction to Machine Learning for Engineers (Foundations and Trends)* (Foundations and Trends in Signal Processing Series). Norwell, MA, USA: Now Publishers, 2018.

[28] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS*, Fort Lauderdale, FL, USA, Apr. 2017, pp. 1–10.

[29] J. Konecný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," Oct. 2016, *arXiv:1610.02527*. [Online]. Available: https://arxiv.org/abs/1610.02527

[30] J. Konecny, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *Proc. NIPS Workshop PMPML*, Barcelona, Spain, Dec. 2016.

[31] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," Apr. 2018, *arXiv:1804.08333*. [Online]. Available: https://arxiv.org/abs/1804.08333

[32] U. O. Oulu. *6G Flagship*. Accessed: Dec. 4, 2018. [Online]. Available: http://www.oulu.fi/6gflagship

[33] Nvidia. *Comparison Chart*. Accessed: Dec. 4, 2018. [Online]. Available: https://www.nvidia.com/object/IO_14605.html

[34] InfiniBand. *Architecture Specification*. Accessed: Dec. 4, 2018. [Online]. Available: https://www.infinibandta.org/ibta-specification/

[35] D. K. Panda. *RDMA-Based Networking Technologies and Middleware for Next-Generation Clusters and Data Centers*. Accessed: Dec. 4, 2018. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2018/files/slides/kbnet/keynote_1.pdf

[36] Google. *Cloud TPU*. Accessed: Dec. 4, 2018. [Online]. Available: https://cloud.google.com/tpu/

[37] L. Wang *et al.*, "Superneurons: Dynamic GPU memory management for training deep neural networks," *ACM Special Interest Group Program. Lang.*, vol. 53, no. 1, pp. 41–53, Feb. 2018.

[38] Qualcomm. *Snapdragon 845 Mobile Platform*. Accessed: Dec. 4, 2018. [Online]. Available: https://www.qualcomm.com/products/snapdragon-845-mobile-platform

[39] GSMArena. *Apple iPhone Xs Max*. Accessed: Dec. 4, 2018. [Online]. Available: https://www.gsmarena.com/apple_iphone_xs_max-9319.php

[40] GSMArena. *Google Pixel 3 XL*. Accessed: Dec. 4, 2018. [Online]. Available: https://www.gsmarena.com/google_pixel_3_xl-9257.php

[41] E. Briman. *Another Look at Alphago vs. Lee Sedol: The Power Angle*. Accessed: Dec. 4, 2018. [Online]. Available: https://www.ceva-dsp.com/ourblog/artificial-intelligence-leaps-forward-mastering-the-ancient-game-of-go

[42] N. Farsad, M. Mao, and A. Goldsmith, "Deep learning for joint source-channel coding of text," in *Proc. ICASSP*, Calgary, AB, Canada, 2018, pp. 2326–2330.

[43] 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.

[44] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for dynamic spectrum access in multichannel wireless networks," in *Proc. IEEE GLOBECOM*, Singapore, Dec. 2017, pp. 1–7.

[45] U. Challita, L. Dong, and W. Saad, "Proactive resource management for LTE in unlicensed spectrum: A deep learning perspective," *IEEE Trans. Wireless Commun.*, vol. 17, no. 7, pp. 4674–4689, Jul. 2018.

[46] P. Popovski, O. Simeone, F. Boccardi, D. Gunduz, and O. Sahin, "Semantic-effectiveness filtering and control for post-5G wireless connectivity," 2019, *arXiv:1907.02441*. [Online]. Available: https://arxiv.org/abs/1907.02441

[47] J. Wang and G. Joshi, "Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms," Aug. 2018, *arXiv:1808.07576*. [Online]. Available: https://arxiv.org/abs/1808.07576

[48] B. Zhang, A. Davoodi, and Y. H. Hu, "Exploring energy and accuracy tradeoff in structure simplification of trained deep neural networks," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 4, pp. 836–848, Dec. 2018.

[49] J. Liu and Q. Zhang, "Offloading schemes in mobile edge computing for ultra-reliable low latency communications," *IEEE Access*, vol. 6, pp. 12825–12837, 2018.

[50] *Feasibility Study on New Services and Markets Technology Enablers, Release-14*, document 3GPP TS 22.891, Sep. 2016.

[51] *Study on New Radio Access Technology Physical Layer Aspects, Release-14*, document 3GPP TS 38.802, Sep. 2017.

[52] *Minimum Requirements Related to Technical Performance for IMT-2020 Radio Interface(s)*, document ITU-R M.2410-0, International Telecommunication Union, Nov. 2017.

[53] M. Iwabuchi, A. Benjebbour, Y. Kishiyama, and Y. Okumura, "Field experiments on 5G ultra-reliable low-latency communication (URLLC)," *NTT DOCOMO Tech. J.*, vol. 20, no. 1, pp. 14–23, Jul. 2018.

[54] *Study on Scenarios and Requirements for Next Generation Access Technologies, Release-15*, document 3GPP TR 38.913, Jun. 2018.

[55] G. Pocovi, K. I. Pedersen, and P. Mogensen, "Joint link adaptation and scheduling for 5G ultra-reliable low-latency communications," *IEEE Access*, vol. 6, pp. 28912–28922, 2018.

[56] A. A. Esswie and K. I. Pedersen, "Opportunistic spatial preemptive scheduling for URLLC and eMBB coexistence in multi-user 5G networks," *IEEE Access*, vol. 6, pp. 38451–38463, 2018.

[57] Z. Wu, F. Zhao, and X. Liu, "Signal space diversity aided dynamic multiplexing for eMBB and URLLC traffics," in *Proc. 3rd IEEE Int. Conf. Comput. Commun. (ICCC)*, Chengdu, China, Dec. 2017, pp. 1396–1400.

[58] S. Lien, S. Hung, D. Deng, and Y. J. Wang, "Efficient ultra-reliable and low latency communications and massive machine-type

communications in 5G new radio," in *Proc. IEEE GLOBECOM*, Singapore, Dec. 2017, pp. 1–7.

[59] L. Zhao, X. Chi, and Y. Zhu, "Martingales-based energy-efficient D-ALOHA algorithms for MTC networks with delay-insensitive/URLLC terminals co-existence," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1285–1298, Apr. 2018.

[60] W. Guan, X. Wen, L. Wang, Z. Lu, and Y. Shen, "A service-oriented deployment policy of end-to-end network slicing based on complex network theory," *IEEE Access*, vol. 6, pp. 19691–19701, Apr. 2018.

[61] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view," *IEEE Access*, vol. 6, pp. 55765–55779, 2018.

[62] *Service Requirements for the 5G System; Stage 1, Release-14*, document 3GPP TS 22.261, Sep. 2018.

[63] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, "Intelligent manufacturing in the context of industry 4.0: A review," *Engineering*, vol. 3, no. 5, pp. 616–630, 2017.

[64] B. Liu, D. Jia, K. Lu, D. Ngoduy, J. Wang, and L. Wu, "A joint control–communication design for reliable vehicle platooning in hybrid traffic," *IEEE Trans. Veh. Technol.*, vol. 66, no. 10, pp. 9394–9409, Oct. 2017.

[65] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Communications and control for wireless drone-based antenna array," *IEEE Trans. Commun.*, vol. 67, no. 1, pp. 820–834, Sep. 2018.

[66] E. Yanmaz, S. Yahyanejad, B. Rinner, H. Hellwagner, and C. Bettstetter, "Drone networks: Communications, coordination, and sensing," *Ad Hoc Netw.*, vol. 68, pp. 1–15, Jan. 2018.

[67] M. Aljehani and M. Inoue, "Communication and autonomous control of multi-UAV system in disaster response tasks," in *Proc. AAMAS*, Stockholm, Sweden, vol. 74, Jul. 2018, pp. 123–132.

[68] E. Joelianto, "Networked control systems: Time delays and robust control design issues," in *Proc. 2nd ICA*, Bandung, Indonesia, Nov. 2011, pp. 16–25.

[69] T. K. Vu, M. Bennis, M. Debbah, M. Latva-Aho, and C. S. Hong, "Ultra-reliable communication in 5G mmWave networks: A risk-sensitive approach," *IEEE Commun. Lett.*, vol. 22, no. 4, pp. 708–711, Apr. 2018.

[70] E. Akyol, U. Mitra, and A. Nayyar, "Controlled sensing and event based communication for remote estimation," in *Proc. 52nd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2014, pp. 545–549.

[71] J. Li, P. Zeng, X. Zong, M. Zheng, and X. Zhang, "Communication and control co-design for wireless sensor networked control systems," in *Proc. 11th World Congr. Intell. Control Automat. (WCICA)*, Jun. 2014, pp. 156–161.

[72] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Dec. 1999, pp. 1057–1063.

[73] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Cambridge, MA, USA: Harvard Univ., 1975.

[74] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[75] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, Oct. 2014, pp. 1724–1734.

[76] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proc. IJCAI*, Barcelona, Spain, Jul. 2011, pp. 1237–1242.

[77] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proc. 26th Annu. Int. Conf. Mach. Learn. (ICML)*, New York, NY, USA, Jun. 2009, pp. 609–616.

[78] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[79] C. D. Manning and H. Schuetze, *Foundations of Statistical Natural Language Processing*, 1st ed. Cambridge, MA, USA: MIT Press, Jun. 1999.

[80] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," in *Proc. 34th ICML*, vol. 70, Sydney, NSW, Australia, Aug. 2017, pp. 214–223.

[81] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Comput. Surv.*, vol. 52, no. 4, Feb. 2018, Art. no. 65.

[82] T. G. Dietterich, "Steps toward robust artificial intelligence," *AI Mag.*, vol. 38, pp. 3–24, Oct. 2017.

[83] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[84] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control, Signals Syst.*, vol. 2, no. 4, pp. 303–314, 1989.

[85] B. Hanin, "Universal function approximation by deep neural nets with bounded width and ReLU activations," Aug. 2017, *arXiv:1708.02691*. [Online]. Available: https://arxiv.org/abs/1708.02691

[86] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *Proc. AISTATS*, San Diego, CA, USA, May 2015, pp. 192–204.

[87] S. Becker, Y. Zhang, and A. Lee, "Geometry of energy landscapes and the optimizability of deep neural networks," Aug. 2018, *arXiv:1808.00408*. [Online]. Available: https://arxiv.org/abs/1808.00408

[88] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *Proc. ITW*, Jerusalem, Israel, May 2015, pp. 1–5.

[89] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," Apr. 2017, *arXiv:1703.00810*. [Online]. Available: https://arxiv.org/abs/1703.00810

[90] C.-W. Huang and S. S. S. Narayanan, "Flow of Renyi information in deep neural networks," in *Proc. IEEE MLSP*, Vietri sul Mare, Italy, Sep. 2016, pp. 1–6.

[91] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Hoboken, NJ, USA: Wiley, 2006.

[92] A. M. Saxe *et al.*, "On the information bottleneck theory of deep learning," in *Proc. ICLR*, Vancouver, BC, Canada, May 2018, pp. 1–27.

[93] A. Kolchinsky, B. D. Tracey, and S. V. Kuyk, "Caveats for information bottleneck in deterministic scenarios," in *Proc. ICLR*, New Orleans, LA, USA, May 2019, pp. 1–23.

[94] R. A. Amjad, and B. C. Geiger, "Learning representations for neural network-based classification using the information bottleneck principle," Feb. 2018, *arXiv:1802.09766*. [Online]. Available: https://arxiv.org/abs/1802.09766

[95] S. Hochreiter and J. Schmidhuber, "Flat minima," *Neural Comput.*, vol. 9, no. 1, pp. 1–42, 1997.

[96] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, "Sharp minima can generalize for deep nets," in *Proc. ICML*, Sydney, NSW, Australia, Jul. 2017, pp. 1019–1028.

[97] P. Chaudhari *et al.*, "Entropy-SGD: Biasing gradient descent into wide valleys," in *Proc. ICLR*, Toulon, France, Apr. 2017, pp. 1–19.

[98] L. Wu, Z. Zhu, and W. E, "Towards understanding generalization of deep learning: Perspective of loss landscapes," Nov. 2017. *arXiv:1706.10239*. [Online]. Available: https://arxiv.org/abs/1706.10239

[99] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape OF neural nets," in *Proc. Adv. NIPS*, Montreal, QC, Canada, Dec. 2018, pp. 6389–6399.

[100] F. Draxler, K. Veschgini, M. Salmhofer, and F. A. Hamprecht, "Essentially no barriers in neural network energy landscape," in *Proc. ICML*, Stockholm, Sweden, Jul. 2018, pp. 1309–1318.

[101] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. 25th NIPS*, vol. 1, Dec. 2012, pp. 1223–1231.

[102] M. A. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized stochastic gradient descent," in *Proc. NIPS*, Vancouver, BC, Canada, Dec. 2010, pp. 2595–2603.

[103] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," in *Proc. ICLR*, San Juan, Puerto Rico, May 2016, pp. 1–10.

[104] P. H. Jin, Q. Yuan, F. Iandola, and K. Keutzer, "How to scale distributed deep learning?" in *Proc. NIPS Workshop SysML*, Barcelona, Spain, Dec. 2016, pp. 1–16.

[105] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Proc. NIPS*, Long Beach, CA, USA, Dec. 2017.

[106] A. Elgabli, M. Bennis, and V. Aggarwal, "Communication-efficient decentralized machine learning framework for 5G and beyond," in *Proc. Globecom*, Hawaii, HI, USA, Dec. 2019, pp. 1–6.

[107] M. Wang, C.-C. Huang, and J. Li, "Supporting very large models using automatic dataflow graph partitioning," in *Proc. 14th EuroSys Conf.*, Dresden, Germany, Mar. 2019, pp. 26:1–26:17.

[108] F. Schuiki, M. Schaffner, F. K. Gürkaynak, and L. Benini, "A scalable near-memory architecture for training deep neural networks on large in-memory datasets," *IEEE Trans. Comput.*, vol. 68, no. 4, pp. 484–497, Apr. 2019.

[109] S. Zhang, A. Choromanska, and Y. LeCun, "Deep learning with elastic averaging SGD," in *Proc. NIPS*, Montréal, QC, Canada, Dec. 2015, pp. 685–693.

[110] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ISCA*, Toronto, ON, Canada, Jun. 2017, pp. 1–12.

[111] 7-CPU. *Intel Skylake X*. Accessed: Dec. 4, 2018. [Online]. Available: https://www.7-cpu.com/cpu/skylake_x.html

[112] AnandTech. *Intel SSD DC P3700 Review*. Accessed: Dec. 4, 2018. [Online]. Available: https://www.anandtech.com/show/8104/intel-ssd-dc-p3700-review-the-pcie-ssd-transition-begins-with-nvme

[113] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. ICLR*, San Juan, Puerto Rico, May 2016, pp. 1–14.

[114] S. Narang *et al.*, "Mixed precision training," in *Proc. ICLR*, Vancouver, BC, Canada, May 2018, pp. 1–12.

[115] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[116] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proc. ICML*, Atlanta, GA, USA, vol. 28, May 2013, pp. 1058–1066.

[117] L. Theis, I. Korshunova, A. Tejani, and F. Huszar, "Faster gaze prediction with dense networks and Fisher pruning," Jul. 2018, *arXiv:1801.05787*. [Online]. Available: https://arxiv.org/abs/1801.05787

[118] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. NIPS Workshop Deep Learn.*, Montréal, QC, Canada, Dec. 2014, pp. 1–9.

[119] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.

[120] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," in *Proc. EuCNC*, Oulu, Finland, Jun. 2017, pp. 1–6.

[121] C.-F. Liu, M. Bennis, and H. V. Poor, "Latency and

reliability-aware task offloading and resource allocation for mobile edge computing," in *Proc. IEEE Globecom Workshops*, Singapore, Dec. 2017, pp. 1–7.

[122] M. S. Elbamby *et al.*, "Wireless edge computing with latency and reliability guarantees," *Proc. IEEE*, vol. 107, no. 8, pp. 1717–1737, Aug. 2019.

[123] C. Gentry and D. Boneh, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009.

[124] J. Park, S.-L. Kim, and J. Zander, "Tractable resource management with uplink decoupled millimeter-wave overlay in ultra-dense cellular networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 6, pp. 4362–4379, Jun. 2016.

[125] R. Meir and T. Zhang, "Generalization error bounds for Bayesian mixture algorithms," *J. Mach. Learn. Res.*, vol. 4, pp. 839–860, Dec. 2003.

[126] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[127] R. M. Neal, "Bayesian learning for neural networks," Ph.D. dissertation, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 1995.

[128] J. Fink, A. Ribeiro, and V. Kumar, "Robust control of mobility and communications in autonomous robot teams," *IEEE Access*, vol. 1, pp. 290–309, 2013.

[129] C. Metz. *Machine Learning Invades the Real World on Internet Balloons*. Accessed: Dec. 4, 2018. [Online]. Available: https://www.wired.com/2017/02/machine-learning-drifting-real-world-internet-balloons

[130] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *Proc. ICML*, Bellevue, WA, USA, Jun. 2011, pp. 681–688.

[131] G. K. Dziugaite and D. M. Roy, "Entropy-SGD optimizes the prior of a PAC-Bayes bound: Generalization properties of entropy-SGD and data-dependent priors," in *Proc. ICML*, Stockholm, Sweden, Jul. 2018.

[132] D. Haussler, "Decision theoretic generalizations of the PAC model for neural net and other learning applications," *Inf. Comput.*, vol. 100, pp. 78–150, Sep. 1992.

[133] J. Langford and M. Seeger, "Bounds for averaging classifiers," Ph.D. dissertation, Dept. Comput. Sci., Carnegie Mellon, Pittsburgh, PA, USA, Jan. 2001.

[134] G. K. Dziugaite and D. M. Roy, "Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameteres than training data," in *Proc. NIPS*, Long Beach, CA, USA, Dec. 2017, pp. 1–14.

[135] D. A. McAllester, "Some PAC-Bayesian theorems," *Mach. Learn.*, vol. 37, no. 3, pp. 355–363, Dec. 1999.

[136] M. Seeger, "PAC-Bayesian generalisation error bounds for Gassian process classification," *J. Mach. Learn. Res.*, vol. 3, pp. 233–269, Oct. 2002.

[137] E. W. Weisstein. *Fractional Chromatic Number*. Accessed: Dec. 4, 2018. [Online]. Available: http://mathworld.wolfram.com/FractionalChromaticNumber.html

[138] L. Ralaivola, M. Szafranski, and G. Stempfel, "Chromatic PAC-Bayes bounds for non-IID data: Applications to ranking and stationary β-mixing processes," *J. Mach. Learn. Res.*, vol. 11, pp. 1927–1956, Jan. 2010.

[139] B. London, B. Huang, and L. Getoor, "Stability and generalization in structured prediction," *J. Mach. Learn. Res.*, vol. 17, pp. 1–52, Apr. 2016.

[140] G. K. Dziugaite and D. M. Roy, "Data-dependent PAC-Bayes priors via differential privacy," in *Proc. NIPS*, Montreal, QC, Canada, Dec. 2018, pp. 8440–8450.

[141] M. Haenggi, "The meta distribution of the SIR in Poisson bipolar and cellular networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 4, pp. 2577–2589, Apr. 2016.

[142] M. Angjelichinoski, K. F. Trillingsgaard, and P. Popovski, "A statistical learning approach to ultra-reliable low latency communication," Sep. 2018, *arXiv:1809.05515*. [Online]. Available: https://arxiv.org/1809.05515

[143] R. Rockafellar and S. Uryasev, "Conditional value-at-risk for general loss distributions," *J. Banking Finance*, vol. 26, no. 7, pp. 1443–1471, Jul. 2002.

[144] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, Sydney, NSW, Australia, vol. 70, Aug. 2017, pp. 449–458.

[145] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI*, San Francisco, CA, USA, Feb. 2017, pp. 1–14.

[146] H. Markowitz, "Portfolio selection," *J. Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[147] A. Tamar, Y. Glassner, and S. Mannor, "Optimizing the CVaR via sampling," in *Proc. 29th AAAI*, Austin, TX, USA, Jan. 2015, pp. 2993–2999.

[148] S. Stanko, "Risk-averse distributional reinforcement learning," M.S. thesis, Dept. Comput. Sci., Czech Tech. Univ., Prague, Czechia, May 2018.

[149] E. Vigonotto and S. Engelke, "Extreme value theory for open set classification—GPD and GEV classifiers," Aug. 2018, *arXiv:1808.09902*. [Online]. Available: https://arxiv.org/1808.09902

[150] T.-W. Weng *et al.*, "Evaluating the robustness of neural networks: An extreme value theory approach," in *Proc. ICLR*, Vancouver, BC, Canada, Jan. 2018, pp. 1–18.

[151] E. M. Rudd, L. P. Jain, W. J. Scheirer, and T. E. Boult, "The extreme value machine," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 3, pp. 762–768, Mar. 2018.

[152] G. Peyré and M. Cuturi, "Computational optimal transport," *Found. Trends Mach. Learn.*, vol. 11, pp. 355–607, Feb. 2019.

[153] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," in *Proc. Adv. Neural Inf. Process. Syst.*, Jun. 2013, pp. 2292–2300.

[154] R. Sinkhorn, "A relationship between arbitrary positive matrices and doubly stochastic matrices," *Ann. Math. Statist.*, vol. 35, no. 2, pp. 876–879, 1964.

[155] P. Khadivi, R. Tandon, and N. Ramakrishnan, "Flow of information in feed-forward deep neural networks," in *Proc. ICCI-CC*, Berkeley, CA, USA, Jul. 2018, pp. 166–173.

[156] G. Chechik, A. Globerson, N. Tishby, and Y. Weiss, "Information bottleneck for Gaussian variables," *J. Mach. Learn. Res.*, vol. 6, pp. 165–188, Jan. 2005.

[157] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *Proc. 48th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, Washington, DC, USA, Oct. 2007, pp. 94–103.

[158] F. Liu, "Generalized Gaussian mechanism for differential privacy," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 4, pp. 747–756, Jun. 2018.

[159] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2325–2383, Oct. 1998.

[160] J.-M. Lasry and P.-L. Lions, "Mean field games," *Jpn. J. Math.*, vol. 2, no. 1, pp. 229–260, 2007.

[161] A. Bensoussan, J. Frehse, and P. Yam, *Mean Field Games and Mean Field Type Control Theory* (Springer Briefs in Mathematics), 1st ed. New York, NY, USA: Springer-Verlag, 2013.

[162] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett, "Functional mechanism: Regression analysis under differential privacy," *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1364–1375, 2012.

[163] B. Sau and V. Balasubramanian, "Deep model compression: Distilling knowledge from noisy teachers," Nov. 2016, *arXiv:1610.09650*. [Online]. Available: https://arxiv.org/abs/1610.09650

[164] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," in *Proc. 37th Annu. Allerton Conf. Commun. Control Comput.*, Monticello, IL, USA, Sep. 1999, pp. 368–377.

[165] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Massive UAV-to-ground communication and its stable movement control: A mean-field approach," in *Proc. IEEE SPAWC*, Kalamata, Greece, Jun. 2018, pp. 1–5.

[166] J. Park, S. Jung, S.-L. Kim, M. Bennis, and M. Debbah, "User-centric mobility management in ultra-dense cellular networks under spatio-temporal dynamics," in *Proc. IEEE GLOBECOM*, Washington, DC, USA, Dec. 2016, pp. 1–6.

[167] H. Kim, J. Park, M. Bennis, S.-L. Kim, and M. Debbah, "Ultra-dense edge caching under spatio-temporal demand and network dynamics," in *Proc. IEEE ICC*, Paris, France, May 2017, pp. 1–7.

[168] W. E, J. Han, and A. Jentzen, "Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations," *Commun. Math. Statist.*, vol. 5, pp. 349–380, Dec. 2017.

[169] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," in *Proc. 35th ICML*, vol. 80, Jul. 2018, pp. 5571–5580.

[170] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Long Beach, CA, USA: Curran Associates, 2017, pp. 4424–4434.

[171] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th ICML*, Sydney, NSW, Australia, 2017, pp. 1126–1135.

[172] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Commun. Lett.*, to be published.

[173] P. Chaudhari, A. Oberman, S. Osher, S. Soatto, and G. Carlier, "Deep relaxation: Partial differential equations for optimizing deep neural networks," in *Proc. Asilomar*, Pacific Grove, CA, USA, Nov. 2017.

[174] Y. Zhang and D.-Y. Yeung, "A regularization approach to learning task relationships in multitask learning," *ACM Trans. Knowl. Discovery Data*, vol. 8, pp. 12:1–12:31, Jun. 2013.

[175] H. Seo, J. Park, M. Bennis, and W. Choi, "Consensus-before-talk: Distributed dynamic spectrum access via distributed spectrum ledger technology," in *Proc. DySPAN*, Seoul, South Korea, Oct. 2018, pp. 1–7.

[176] H. Seo, J. Park, M. Bennis, and W. Choi, "Communication and consensus co-design for low-latency and reliable industrial IoT systems," Jul. 2019, *arXiv:1907.08116*. [Online]. Available: https://arxiv.org/abs/1907.08116

[177] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. NIPS*, Lake Tahoe, NV, USA, Dec. 2013, pp. 315–323.

[178] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *Proc. 31st Mach. Learn. Res.*, Bejing, China, vol. 32, Jun. 2014, pp. 1000–1008.

[179] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-IID data," Jun. 2018, *arXiv:1806.00582*. [Online]. Available: https://arxiv.org/abs/1806.00582

[180] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Communication-efficient on-device machine learning: Federated distillation and augmentation under non-IID private data," presented at the NeurIPS Workshop MLPCD Montréal, QC, Canada, Dec. 2018. [Online]. Available: https://arxiv.org/abs/1811.11479

[181] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton, "Large scale distributed neural network training through online distillation," Apr. 2018, *arXiv:1804.03235*. [Online]. Available: https://arxiv.org/abs/1804.03235

[182] L. J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proc. NIPS*, Montreal, QC, Canada,

Dec. 2014, pp. 2654–2662.

[183] A. Elgabli, J. Park, A. S. Bedi, M. Bennis, and V. Aggarwal, "GADMM: Fast and communication efficient framework for distributed machine learning," pp. 1–33, Aug. 2019, *arXiv:1909.00047*. [online]. Available: https://arxiv.org/abs/1909.00047

[184] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," Sep. 2018, *arXiv:1809.07857*. [Online]. Available: https://arxiv.org/abs/1809.07857

[185] S. Srinivas and F. Fleuret, "Knowledge transfer with Jacobian matching," in *Proc. ICML*, Stockholm, Sweden, Jul. 2018, pp. 1–11.

[186] J. Park *et al.*, "Distilling on-device intelligence at the network edge," Aug. 2019, *arXiv:1908.05895*. [Online]. Available: https://arxiv.org/abs/1908.05895

[187] Z. Jiang, A. Balu, C. Hegde, and S. Sarkar, "Collaborative deep learning in fixed topology networks," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. New York, NY, USA: Curran Associates, 2017, pp. 5904–5914.

[188] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," in *Proc. ICLR*, Vancouver, BC, Canada, May 2018, pp. 1–12.

[189] H. Cha, J. Park, H. Kim, S.-L. Kim, and M. Bennis, "Federated reinforcement distillation with proxy experience memory," Jul. 2019, *arXiv:1907.06536*. [Online]. Available: https://arxiv.org/abs/1907.06536

[190] V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: https://arxiv.org/abs/1312.5602

[191] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Proc. Adv. NIPS*, Long Beach, CA, USA, Dec. 2017, pp. 1709–1720.

[192] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," in *Proc. Adv. NIPS*, Montreal, QC, Canada, Dec. 2018, pp. 4447–4458.

[193] T. Chen, G. Giannakis, T. Sun, and W. Yin, "LAG: Lazily aggregated gradient for communication-efficient distributed learning," in *Proc. Adv. NIPS*, Montreal, QC, Canada, Dec. 2018, pp. 5050–5060.

[194] S. Wang *et al.*, "Adaptive federated learning in resource constrained edge computing systems," Feb. 2019, *arXiv:1804.05271*. [Online]. Available: https://arxiv.org/abs/1804.05271

[195] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," Aug. 2019, *arXiv:1908.07873*. [Online]. Available: https://arxiv.org/abs/1908.07873

[196] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, p. 12, Feb. 2019.

[197] S. L. Smith and Q. V. Le, "A Bayesian perspective on generalization and stochastic gradient descent," in *Proc. ICLR*, Vancouver, BC, Canada, Oct. 2018, pp. 1–13.

[198] C. D. Sa *et al.*, "High-accuracy low-precision training," Mar. 2018, *arXiv:1803.03383*. [Online]. Available: https://arxiv.org/abs/1803.03383

[199] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for mobile edge computing," in *Proc. 16th Int. Symp. Modeling Optim. Mobile, Ad Hoc, Wireless Netw. (WiOpt)*, Shanghai, China, May 2018, pp. 1–6.

[200] J. McInerney *et al.*, "Explore, exploit, and explain: Personalizing explainable recommendations with bandits," in *Proc. 12th ACM RecSys*, Vancouver, BC, Canada, Oct. 2018, pp. 31–39.

[201] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. CVPR*, Honolulu, HI, USA, Jul. 2017, pp. 5687–5695.

[202] D. Lee, D. Ahm, T. Kim, P. I. Chuang, and J.-J. Kim, "Viterbi-based pruning for sparse matrix with fixed and high index compression ratio," in *Proc. ICLR*, Vancouver, BC, Canada, May 2018, pp. 1–16.

[203] W. Wen *et al.*, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. New York, NY, USA: Curran Associates, 2017, pp. 1509–1519.

[204] C. Hardy, E. Le Merrer, and B. Sericola, "MD-GAN: Multi-discriminator generative adversarial networks for distributed datasets," Nov. 2018, *arXiv:1811.03850*. [Online]. Available: https://arxiv.org/abs/1811.03850

[205] Q. Hoang, T. D. Nguyen, T. Le, and D. Q. Phung, "Multi-generator generative adversarial nets," Oct. 2017, *arXiv:1708.02556*. [Online]. Available: https://arxiv.org/abs/1708.02556

[206] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," Dec. 2018, *arXiv:1812.00564*. [Online]. Available: https://arxiv.org/abs/1812.00564

[207] P. Vepakomma, O. G. A. Dubey, and R. Raskar, "Reducing leakage in distributed deep learning for sensitive health data," in *Proc. ICLR*, New Orleans, FL, USA, May 2019, pp. 1–6.

[208] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," Jun. 2018, *arXiv:1806.07366*. [Online]. Available: https://arxiv.org/abs/1806.07366

[209] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications," Jul. 2018, *arXiv:1807.08127*. [Online]. Available: https://arxiv.org/abs/1807.08127

[210] G. Montavon, K.-R. Müller, and M. Cuturi, "Wasserstein training of restricted Boltzmann machines," in *Proc. NIPS*, Barcelona, Spain, Dec. 2016, pp. 3718–3726.

[211] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Dec. 4, 2018. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[212] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Proc. IEEE P2P*, Trento, Italy, Sep. 2013, pp. 1–10.

[213] Sierra Wireless, "Coverage analysis of LTE-M category-M1," Sierra Wireless, Richmond, BC, Canada, White Paper, Jan. 2017.

[214] M. Mirza and S. Osindero, "Conditional generative adversarial nets," Nov. 2014, *arXiv:1411.1784*. [Online]. Available: https://arxiv.org/abs/1411.1784

[215] E. Jeong, S. Oh, J. Park, H. Kim, M. Bennis, and S.-L. Kim, "Multi-hop federated private data augmentation with sample compression," Jul. 2019, *arXiv:1907.06426*. [Online]. Available: https://arxiv.org/abs/1907.06426

[216] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Edge computing meets millimeter-wave enabled VR: Paving the way to cutting the cord," in *Proc. IEEE WCNC*, Barcelona, Spain, Apr. 2018, pp. 1–6.

[217] A. Elgabli, H. Khan, M. Krouka, and M. Bennis, "Reinforcement learning based scheduling algorithm for optimizing age of information in ultra reliable low latency networks," Nov. 2018, *arXiv:1811.06776*. [Online]. Available: https://arxiv.org/abs/1811.06776

[218] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Mach. Learn. Res. (ICML)*, New York, NY, USA, vol. 48, Jun. 2016, pp. 1928–1937.

[219] H. Shiri, J. Park, and M. Bennis, "Massive autonomous UAV path planning: A neural network based mean-field game theoretic approach," presented at the GLOBECOM, May 2019.

ABOUT THE AUTHORS

Jihong Park (Member, IEEE) received the B.S. and Ph.D. degrees from Yonsei University, Seoul, South Korea, in 2009 and 2016, respectively.

From 2016 to 2017, he was a Postdoctoral Researcher with Aalborg University, Aalborg, Denmark. He was a Visiting Researcher with The Hong Kong Polytechnic University, Hong Kong, in 2013; the KTH Royal Institute of Technology, Stockholm, Sweden, in 2015; Aalborg University, in 2016; and the New Jersey Institute of Technology, Newark, NJ, USA, in 2017. He is currently a Postdoctoral Researcher with the University of Oulu, Oulu, Finland. From 2020, he will be a Lecturer at the School of Information Technology, Deakin University, Melbourne, VIC, Australia. His current research interests include distributed machine learning and ultrareliable designs in 5G communication systems and beyond.

Dr. Park received the 2014 IEEE GLOBECOM Student Travel Grant, the 2014 IEEE Seoul Section Student Paper Contest Bronze Prize, and the 6th IDIS-ETNEWS (The Electronic Times) Paper Contest Award sponsored by the Ministry of Science, ICT and Future Planning of South Korea.

Sumudu Samarakoon (Member, IEEE) received the B.Sc. degree (Hons.) in electronic and telecommunication engineering from the University of Moratuwa, Moratuwa, Sri Lanka, in 2009, the M.Eng. degree from the Asian Institute of Technology, Khlong Luang, Thailand, in 2011, and the Ph.D. degree in communication engineering from the University of Oulu, Oulu, Finland in 2017.

He is currently with the Centre for Wireless Communications (CWC), University of Oulu, as a Postdoctoral Researcher. His current research interests include heterogeneous networks, small cells, radio resource management, reinforcement learning, and game theory.

Dr. Samarakoon received the Best Paper Award at the European Wireless Conference and the Excellence Awards for innovators and the outstanding doctoral student at the Radio Technology Unit, CWC, University of Oulu, in 2016.

Mehdi Bennis is currently an Associate Professor with the Centre for Wireless Communications, University of Oulu, Oulu, Finland, where he is also an Academy of Finland Research Fellow and the Head of the Intelligent Connectivity and Networks/Systems Group (ICON). He has coauthored one book and published more than 200 research articles in international conferences, journals, and book chapters. His current research interests include radio resource management, heterogeneous networks, game theory, and machine learning in 5G networks and beyond.

Dr. Bennis was a recipient of several prestigious awards, including the 2015 Fred W. Ellersick Prize from the IEEE Communications Society, the 2016 Best Tutorial Prize from the IEEE Communications Society, the 2017 EURASIP Best Paper Award for the *Journal on Wireless Communications and Networks*, the All-University of Oulu Award for research, and the 2019 IEEE ComSoc Radio Communications Committee Early Achievement Award. He is an Editor of the IEEE TRANSACTIONS ON COMMUNICATIONS.

Mérouane Debbah (Fellow, IEEE) received the M.Sc. and Ph.D. degrees from the École Normale Supérieure Paris-Saclay, Cachan, France.

He was with Motorola Labs, Saclay, France, from 1999 to 2002, and the Vienna Research Center for Telecommunications, Vienna, Austria, until 2003. From 2003 to 2007, he was an Assistant Professor with the Mobile Communications Department, Institut Eurecom, Sophia Antipolis, France. From 2007 to 2014, he was the Director of the Alcatel-Lucent Chair on Flexible Radio. Since 2007, he has been a Full Professor with the CentraleSupélec, Gif-sur-Yvette, France. Since 2014, he has been the Vice President of the Huawei France Research Center and the Director of the Mathematical and Algorithmic Sciences Lab, Paris, France. He has managed eight EU projects and more than 24 national and international projects. His current research interests include fundamental mathematics, algorithms, statistics, information, and communication sciences research.

Dr. Debbah is a WWRF Fellow and a Membre mrite SEE. He was a recipient of the ERC Grant MORE (Advanced Mathematical Tools for Complex Network Engineering) from 2012 to 2017, the Mario Boella Award in 2005, the IEEE Glavieux Prize Award in 2011, and the Qualcomm Innovation Prize Award in 2012. He received 20 best paper awards, among which the 2007 IEEE GLOBECOM Best Paper Award, the Wi-Opt 2009 Best Paper Award, the 2010 Newcom++ Best Paper Award, the WUN CogCom Best Paper 2012 and 2013 Awards, the 2014 WCNC Best Paper Award, the 2015 ICC Best Paper Award, the 2015 IEEE Communications Society Leonard G. Abraham Prize, the 2015 IEEE Communications Society Fred W. Ellersick Prize, the 2016 IEEE Communications Society Best Tutorial Paper Award, the 2016 European Wireless Best Paper Award, the 2017 Eurasip Best Paper Award, the 2018 IEEE Marconi Prize Paper Award, the 2019 IEEE Communications Society Young Author Best Paper Award, and the Valuetools 2007, Valuetools 2008, CrownCom 2009, Valuetools 2012, SAM 2014, and 2017 IEEE Sweden VT-COM-IT Joint Chapter Best Student Paper Awards. He was an Associate Area Editor and a Senior Area Editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING from 2011 to 2013 and from 2013 to 2014, respectively. He is an Associate Editor-in-Chief of the journal *Random Matrix: Theory and Applications*.