On-device Learning Systems for Edge Intelligence: A Software and Hardware Synergy Perspective

Qihua Zhou, *Student Member, IEEE*, Zhihao Qu, *Member, IEEE*, Song Guo, *Fellow, IEEE*, Boyuan Luo, Jingcai Guo, *Member, IEEE*, Zhenda Xu, and Rajendra Akerkar

Abstract—Modern machine learning (ML) applications are often deployed in the cloud environment to exploit the computational power of clusters. However, this in-cloud computing scheme cannot satisfy the demands of emerging edge intelligence scenarios, including providing personalized models, protecting user privacy, adapting to real-time tasks and saving resource cost. In order to conquer the limitations of conventional in-cloud computing, it comes the rise of on-device learning, which makes the end-to-end ML procedure totally on user devices, without unnecessary involvement of the cloud. In spite of the promising advantages of on-device learning, implementing a high-performance on-device learning system still faces with many severe challenges, such as insufficient user training data, backward propagation blocking and limited peak processing speed. Observing the substantial improvement space in the implementation and acceleration of on-device learning systems, we intend to present a comprehensive analysis of the latest research progress and point out potential optimization directions from the system perspective. This survey presents a software and hardware synergy of on-device learning techniques, covering the scope of model-level neural network design, algorithm-level training optimization and hardware-level instruction acceleration. We hope this survey could bring fruitful discussions and inspire the researchers to further promote the field of edge intelligence.

Index Terms—On-device Learning, System Implementation, Neural Network Design, Training Algorithm Optimization, Hardware Acceleration.

I. INTRODUCTION

MACHINE learning (ML) techniques have greatly promote the development of intelligent applications, including computer vision [1, 2], natural language processing [3], big data analytics [4] and robotic process automation [5]. In practice, the deployment of these applications often resort to the model training with large-scale datasets in the cloud environment [6], requiring huge demands of computational resources.

Qihua Zhou is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. E-mail: csqzhou@comp.polyu.edu.hk.

Zhihao Qu is with the School of Computer and Information, Hohai University, and Department of Computing, The Hong Kong Polytechnic University. E-mail: quzhihao@hhu.edu.cn.

Song Guo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, and The Hong Kong Polytechnic University Shenzhen Research Institute. E-mail: song.guo@polyu.edu.hk.

Boyuan Luo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. E-mail: boyuan.luo@connect.polyu.hk.

Jingcai Guo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. E-mail: cscjguo@comp.polyu.edu.hk.

Zhenda Xu is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. E-mail: jackal.xu@connect.polyu.hk.

Rajendra Akerkar is with the Information Technology of Western Norway Research Institute, Sogndal, Norway. E-mail: rak@vestforsk.no.

Corresponding authors: Song Guo and Zhihao Qu.

Although many in-cloud learning approaches have been proposed, they may not well match the emerging trend that *enables edge-intelligent features running on personal devices by using user local data only* [7, 8]. In this scenario, the main drawbacks of in-cloud learning can be summarized into three aspects. The first is the privacy and security issue. As the intermediate training results and model parameters need to be transmitted through the network and stored in the cloud, the user data containing sensitive information can be easily eavesdropped and user privacy will be exposed to the cloud carrier [9]. The second problem is that the in-cloud learning cannot provide personalized model for the user as conventional training procedure aims to provide a unified model by aggregating the results from plenty of workers, *i.e.*, distributed training in a data-parallel manner [10]. At last, although it is possible to conduct the application processing in the cloud and fetch the final result to the user to meet the personalized requirement, the data transmission may yield an unacceptable network transmission time, especially in the bandwidth-limited environment. This high latency will hamper the real-time model update and lead to a low processing throughput [11].

Observing the limitations of conventional in-cloud learning, it comes the rise of on-device learning, which handles the end-to-end ML training totally on device [12]. Generally, the on-device learning applications are often deployed on the edge devices. Thus, on-device learning also refers to the *edge intelligence* [13] sometimes. The basic objective of on-device learning is to swiftly produce the personalized model with good quality in the resource-constrained environment, *e.g.*, the Tensorflow Lite [14] and PyTorch Mobile [15].

However, achieving this target requires a co-design of learning algorithms and system implementation, where several challenges need to be addressed. First, an edge device usually owns few user data [16], which are not sufficient enough and may lead to model overfitting [17]. Besides, the backward propagation for model update may be blocked as not all the neurons or layers are updatable in the on-device environment [18]. Moreover, the peak processing speed is often restricted by the operating system because the devices are not built for running at full speed continuously in a long time, *i.e.*, the generated heat will slow down the processors and exhaust the limited battery. As shown in Fig. 1, we present the motivation that shifts in-cloud learning to on-device learning, and point out the potential challenges in system implementation.

Consequently, designing a high-performance on-device system is not easy in practice. In this survey, we intend to guide

Fig. 1: Systems for edge intelligence: from in-cloud computing to on-device learning.

the readers to build efficient on-device learning frameworks. As to the optimization of system-level performance, we also give deep insights of on-device learning acceleration. Note that our survey stands in the perspective of system implementation in a full stack, contains the scope of model-level neural network design (§III), algorithm-level training optimization (§IV) and hardware-level instruction acceleration (§V). For reading convenience, we use Fig. 2 to illustrate the organization of research topics and present the overview of system architecture. Overall, the topics of model-level neural network design and algorithm-level training optimization are two major directions to optimize the performance of edge intelligence applications. They collaborate together to build the upper on-device learning software and extract the most significant metrics (e.g., memory footprint, processing speed and prediction accuracy) for the implementation of underlying hardware. The topic of hardware-level instruction acceleration follows the requirements from the learning software and provides domain-specific accelerators to fully improve the system efficiency. This kind of software and hardware synergy serves as a guide for the design of on-device learning systems.

Motivated by these realistic challenges of on-device learning, this survey follows the latest research progress and covers comprehensive optimization topics for the system implementation and acceleration. In order to adapt to the resource-constrained environment, model compression technologies can be used to alleviate the computational pressure, including network-aware parameter pruning (§III-A) and knowledge distillation (§III-B). Also, as to requirements of addressing insufficient user data and enabling swift training, the model fine-tuning approaches are discussed in §III-C. On top of the model-level design of neural networks, we also discuss how to accelerate the learning speed. The data-adaptive regularization methods in §IV-A aim at matching the optimization function with small-scale training data on user devices, so as to avoid model overfitting and improve the convergence efficiency. Besides, we discuss the quantization techniques for inference

Fig. 2: The organization of survey scope and the overview of system architecture.

(§IV-B2) to reduce the representation bit width of the numeric values. We also introduce the quantization into model training (§IV-B3) to fundamentally reduce the computational overhead while not degrading the model quality. Moreover, in order to fully exploit the algorithm-level optimization, this survey also introduces how to use underlying hardware to accelerate the learning speed, including the embedded memory controlling (§V-A), dedicated computational primitives (§V-B), low-level instructions (§V-C), and MIMO-based communication and coding (§V-D).

Apart from the technical discussion, we also evaluate the

effect of applying different methods on system performance (§VI-A) and present the future directions of embedding on-device learning techniques into realistic scenarios, including intelligent medical diagnosis (§VI-B), AI-enhanced motion tracking (§VI-C) and domain-specific acceleration chips (§VI-D).

Overall, this survey focuses on the on-device software and hardware synergy for deploying edge intelligence applications. We present an overview of the survey structure by showing the relationships among research methodologies and related approaches in Table I. Our major contributions are summarized as follows:

- We analyse the limitations of conventional in-cloud computing and reveal that the on-device learning is a promising research direction to meet the requirements of edge intelligence applications.
- We point out the main challenges when deploying on-device learning tasks in real world and guide the researchers how to implement a reliability learning systems, standing in the perspective of a software and hardware synergy.
- We make a comprehensive discussion on state-of-the-art research progress and provide system-level insights of designing on-device learning frameworks, including the topics of neural network design, training algorithm optimization and domain-specific hardware acceleration.
- We summarize the gist of various on-device learning approaches and present potential research directions that can be extended in future work.

We hope this survey could bring fruitful discussions on the on-device learning techniques, inspiring researchers to implement high-efficiency learning systems and promote the development of edge intelligence.

II. BACKGROUND

In this section, we will present the preliminary knowledge of training theorem and introduce the gist of on-device learning.

A. Basic Training Algorithms

We will first introduce the pertinent gradient descent based optimization algorithms. Then, we will discuss the widely-used Deep Learning (DL) techniques using neural networks.

1) *Stochastic Gradient Descent*: In this iterative training scheme, accelerating the processing speed requires the skills of model optimization, which aims to minimize the loss function that evaluates the estimation values and ground-truth values. A common way for this minimization is to use the *Stochastic Gradient Descent* (SGD) [66] method. Briefly, SGD employs the random sampling of the training data. Thus, the corresponding update rule can be formulated as:

$$w_{t+1} = w_t - \eta_t \nabla f_{i,t}(w_t), \quad (1)$$

where i is the data index of the random sampling in the t -th iteration and η_t is the *learning rate* in current iteration. The concept of learning rate is very similar to the step length of different kinds of optimization methods (e.g., the *Steepest Descent* and the *Newton's method*). Note that $f_{i,t}(w_t)$ is the

value the loss function $\mathcal{L}$, which reflects the gap between prediction based on $x_{i,t}$ and ground-truth label $y_{i,t}$. Besides, $R(w_t)$ is the regularization term for shrinking the large ones inside model parameters w_t .

$$f_{i,t}(w_t) = \mathcal{L}(w_t; x_{i,t}; y_{i,t}) + R(w_t). \quad (2)$$

On top of this formulation, we will present the algorithm of SGD via Algorithm 1.

Algorithm 1 Stochastic Gradient Descent Method

Input: Training dataset $\mathbb{X}$

Output: Updated model parameters w_T

```

1: procedure SGD(Training dataset  $\mathbb{X}$ )
2:   Initialize the model parameters;
3:   for  $t = 0, 1, \dots, T - 1$  do
4:     Randomly select a sample  $x_{i,t}$  from  $\mathbb{X}$ ;
5:     Calculate the loss function based on  $x_{i,t}$ ;
6:     Calculate the gradients  $\nabla f_{i,t}(w_t)$  of Eq. (2);
7:     Update the model parameters by using Eq. (1);
8:   end for
9:   return the latest model parameters  $w_T$ ;
10: end procedure

```

According to the algorithm of SGD, it is reasonable to transfer the experience loss function to the average value of the sampling loss function. As SGD employs the bootstrapping (*i.e.*, random sampling with replacement) [67] for gradient calculation, we can obtain the unbiased estimation of standard gradients calculated by all the data, *i.e.*, $\mathbb{E}[\nabla f_{i,t}(w_t)] \leftarrow \nabla f(w_t)$. Meanwhile, as SGD only chooses one sample from the entire dataset, the corresponding computational overhead is significantly reduced. Consequently, it is natural to employ the SGD algorithm for model training.

2) *Mini-batch Stochastic Gradient Descent*: In order to reduce the variance of gradient estimation, there is a variant of SGD, called *Mini-batch Stochastic Gradient Descent* (Mini-batch SGD), which is widely used in modern machine learning applications. In each iteration, the key difference of Mini-batch SGD is to use a batch of samples for calculating loss values and gradients, instead of using one training sample, which can be formally written as:

$$\nabla f_{B_t}(w_t) = \frac{1}{|B_t|} \sum_{i \in B_t} \nabla f_i(w_t), \quad (3)$$

Note that the *batch size* B_t controls the number of samples contained in each batch [68], which is a significant hyper-parameter impacting the training efficiency and convergence speed. Therefore, the update rule is given by:

$$w_{t+1} = w_t - \eta_t \nabla f_{B_t}(w_t), \quad (4)$$

where η_t is the learning rate of the batch. In order to give a clear demonstration of the Mini-batch SGD, we present the corresponding algorithm in Algorithm 2.

TABLE I: Summary of research methodologies and related approaches.

Research Methodology	Approach	Highlights	Advantages	Related Work
Model-level Neural Network Design (§III)	Parameter Pruning (§III-A)	Remove trivial neurons or blocks to simplify the model	Reduce parameter amount	[19–25]
	Knowledge Distillation (§III-B)	Train a lightweight student model with similar performance as the heavy teacher model	Simplify network structure	[26–31]
	Model Fine-tuning (§III-C)	Transfer a pre-trained model to new samples and update a specific portion of layers	Train personalized model with small-scale user data	[32–36]
Algorithm-level Training Optimization (§IV)	Data-adaptive Regularization (§IV-A)	Use regularization term to shrink large parameters	Reduce model complexity	[37–40]
	Numerical Quantization (§IV-B)	Compress tensors by using low-precision data representation	Reduce memory footprint and accelerate arithmetic operations	[41–51]
Hardware-level Instruction Acceleration (§V)	Embedded Memory Controlling (§V-A)	Share limited memory via task-aware scheduling	Improve memory access efficiency	[11, 52–56]
	Dedicated Computational Primitives (§V-B)	Use dedicated computational primitives to handle arithmetic operations	Accelerate training speed	[57–60]
	Low-level Instructions (§V-C)	Design instructions for domain-specific chips	Save computational and energy cost	[61–63]
	MIMO-based Communication (§V-D)	Promote the capability of wireless infrastructures via massive antenna arrays	Alleviate communication bottleneck during training	[64, 65]

Algorithm 2 Mini-batch Stochastic Gradient Descent Method

Input: Training dataset $\mathbb{X}$
Output: Updated model parameters w_T

```

1: procedure SGD(Training dataset  $\mathbb{X}$ )
2:   Initialize the model parameters;
3:   for  $t = 0, 1, \dots, T - 1$  do
4:     Randomly select a mini-batch of samples  $B_t$  from  $\mathbb{X}$ ;
5:     Calculate the loss function based on  $B_t$ ;
6:     Calculate the gradients by using Eq. (3);
7:     Update the model parameters by using Eq. (4);
8:   end for
9:   return the latest model parameters  $w_T$ ;
10: end procedure

```

be compared with the ground-truth labels and we use the loss function to measure the accuracy gap. Aiming at minimizing the loss function based on current data samples, the BP stage usually adopts the SGD-based algorithms and calculates the gradients of all parameters. After updating the model with the gradients, we can continue the training by starting a new iteration.

B. On-device Learning

As modern ML applications often requires a huge amount of computational resources, conventional implementation of the learning approaches usually rely on the power of cloud-based machines deployed in the distributed manner. However, with the rapid development of the hardware processing speed and memory capacity, it is possible to handle some small-scale learning applications by the on-device hardware itself, *e.g.*, the Apple Face ID [69] and Microsoft Flower Recognition [70]. Here, we will introduce the definition of on-device learning, the objective of these applications and the crucial challenges in practice.

1) *Definition:* Generally, on-device learning means handling the ML applications and the corresponding the model training procedure totally on the device, without the involvement of cloud-based machines [12]. As on-device learning is closely related to the rise of edge and mobile devices, it also can be referred as the edge learning or edge intelligence [13] sometimes.

Considering the real-world scenario of edge devices, the on-device learning applications may face with the challenges of expensive computational primitives, low memory capacity and limited I/O bandwidth [71]. Besides, the communication cost is also a crucial issue when the learning applications is deployed in the distributed edge device cluster, because massive network traffic will be yield during the training runtime [72]. Therefore, the edge devices in the scope of on-device learning are often bounded in the resource-constrained environment, where reducing computational overhead, improving hardware efficiency and saving energy cost should be considered for the on-device learning implementation [73].

2) *Objectives:* Building a high-performance on-device learning system requires a clear understanding of why we need on-device learning and what are the advantages on-device learning can bring. Briefly, the objectives of on-device learning can be summarized as follows.

Fig. 3: The workflow of neural network training.

3) *Neural Network Training:* As mentioned above, the theoretically guarantee that the model training can achieve good convergence efficiency comes from the use of gradient descent based optimization method to minimize the loss function. Here, we will discuss the rationale of training neural networks of DL applications. The key is to extract patterns or obtain knowledges from the training data, which are usually in a large scale. These patterns and knowledge can be represented via the key/value pairs stored in the neural network models. We use Fig. 3 to better illustrate the workflow of neural network training. During the training procedure, each iteration can be represented by two stages, *i.e.*, *forward propagation* (FP) and *backward propagation* (BP) stages. The FP stage extracts deep knowledge from the input data by pipelining the intermediate results layer by layer, where the convolutional (CONV) layers yield the feature maps of data samples and the fully connected (FC) layers make the prediction. The final predicted output will

Resource Saving. Due to the resource-constrained environment of the edge devices, a high-performance on-device learning system needs to reduce the resource cost as much as possible, including the hardware-level computation overhead, communication-level network traffic and energy-level consumption. Overall, resource saving is one of the most essential demands to deploy on-device learning [49].

Personalized Model. As on-device learning applications are executed totally based on the user local data, the corresponding learning frameworks cannot be design as usual distributed ML systems in cloud that aim to provide a general model working for an ensemble of tasks. Instead, on-device learning should provide customized models which are tailed to meet the user's preference, *i.e.*, different users own different models based on their needs [11].

Privacy Protection. As the training procedure only happens on the edge devices and the final model contains the sensitive information of user preference, the data privacy should be carefully protected [74]. All the training data should be stored and accessed under the user's control, without any unnecessary data sharing with other machines.

Online Learning. As the user data is generated on the device continuously, the systems should be able to handle the learning procedure online and incrementally re-train the model according to the latest user data, so as to make the customized model up-to-date [75].

Summary. The above four properties closely rely on the co-design of learning algorithms and system implementation, where several challenges need to be addressed. We will discuss them in the next.

3) *Challenges:* Although on-device learning shows promising advantages to deploy modern ML applications in the edge environment, implementing the relevant frameworks in the realistic scenario is not easy. In order to guide the future research of the on-device learning and corresponding system design, we summarize the most significant challenges as follows.

Insufficient User Data. As the data on the edge devices are usually specific to each user and cannot exchange with others due to the privacy concern, the available training data are not sufficient enough to conduct the model training as traditional distributed ML does. This condition is similar to the few-shot learning [16] with relatively few user data. Generally, it is hard to ensure a model with a good generalization based on small-scale training dataset. Therefore, on-device learning cannot simply follow conventional training methods that learn from the scratch. This property requires the systems to learn customized features from the small-scale dataset. We will mainly introduce the techniques of knowledge distillation (§III-B) and model fine-tuning (§III-C) to tackle this challenge.

Backward Propagation Blocking. Backward propagation is the key step to calculate the gradients and update the model parameter. However, when training a deep model on device, conventional chain-rule based gradient calculation may be stuck in some layers and blocks the backward propagation to the next layers. For example, as to the on-device learning using the Core ML framework [76] on iPhone, only the FC and CONV layers can participate in the gradient calculation,

Network	Input Scale	Parameter Memory Footprint (MB)	GFLOPs
AlexNet	227 x 227	233	0.727
SqueezeNet	224 x 224	5	0.837
MobileNet	224 x 224	16	0.579
DenseNet-201	224 x 224	77	4
VGG-16	224 x 224	528	16
VGG-19	224 x 224	548	20
GoogleNet	224 x 224	51	2
ResNet-18	224 x 224	45	2
ResNet-50	224 x 224	98	4
ResNet-101	224 x 224	170	8
ResNet-152	224 x 224	230	11
Inception-V3	299 x 299	91	6

TABLE II: Resource cost of different neural networks.

while the weights based on the operations of batch normalization, embeddings, bias and scales cannot be updated. Besides, models based on RNN, LSTM, GRU are also not well supported. Therefore, the systems need to cover more layers and model properties when deploy on-device learning on mobile phones. This requirement is closely related to the hardware implementation (§V) on different edge platforms. We will cover the aspects of embedded memory, computational primitives, low-level instructions and communication scheme to present a comprehensive discussion.

Limited Peak Speed. Different from the servers in the cloud environment, edge devices, specially the mobile phones, are not built for running at top speed for hours continuously to finish a task. However, the model training procedure requires a huge amount of iterations until the convergence, which will easily generate much heat and invoke the CPU cooling process to protect the expensive hardware. The most common way to restrict the hardware temperature is to control the CPU's clock in a lower frequency, which will definitely slow down the processing speed. Besides, running at peek speed for a long time will exhaust the battery energy of mobile devices [10]. As a result, conducting the learning applications on devices requires more efficient management of the limited resources, where compressing model size and simplifying arithmetic operations are two key steps. We will mainly discuss the topics of parameter pruning for model compression (§III-A), loss regularization for network sparsification (§IV-A) and data quantization for processing acceleration (§IV-B) to reduce the computational overhead.

Summary. The above mentioned challenges are the most critical issues to implement on-device learning systems. In the rest of this survey, we will discuss the related technologies to address these challenges.

III. NEURAL NETWORK DESIGN

Deploying modern learning applications often yield huge computational demands, which may readily exceed the available resources on devices, including computational primitives, memory capacity, I/O bandwidth, network connection and data storage. As shown in Table II, we compare the resource cost of different neural networks when deploying the models in practice. Therefore, optimizing the network structures and compressing model size are promising research topics for the

implementation on-device learning systems. Besides, different from traditional in-cloud ML applications that may not need to consider the consumed energy during the task execution, on-device learning applications are very sensitive to the energy cost as the limited battery equipped by the devices cannot maintain the long-time execution of the jobs. All of these challenges will become the implementation bottleneck to deploy on-device learning applications, requiring researchers to elaborate the design of corresponding on-device learning frameworks. In this section, we will discuss how to conquer above challenges from the perspective of designing high-efficiency neural networks, which contains the following three parts: (1) network-aware parameter pruning, (2) knowledge distillation, and (3) fine-tune training. These three aspects can be regarded as the approaches for model compression, which can be adopted to reduce the hardware-level resource cost.

A. Network-aware Parameter Pruning

Deploying modern ML applications often depends on complex neural networks with huge mode size, parameter amounts and matrix operations, which may easily exceed the available resource capacity of the mobile devices. Consequently, decreasing the computational overhead of the model is one of the most important issues in the acceleration and implementation of the on-device learning. A natural idea to achieve this goal is to reduce the model complexity of the neural network. As a result, it comes the parameter pruning [77] method to simplify the original “elephant” models.

1) *Pruning Steps*: The workflow of designing the parameter pruning algorithms mainly contains five steps: (1) analysing the sparsity of the original neural network and checking whether it is feasible for model compression, (2) inspecting each neuron by measuring the update significance to the entire model, (3) removing the trivial neurons that holds the insignificant parameters (*i.e.*, weights and activations) from the network, (4) re-tuning the compressed model in a fine-grained layer-wise manner and making the final output adapt to the corresponding objective function, (5) checking the model sparsity and keeping pruning the redundant parameters.

Among these steps, correctly measuring the network sparsity is the most fundamental issue. Therefore, we will discuss the sparsity definition first. Sparsity is used to inspect the parameter redundancy of a model by checking the portion of elements with zero or near zero values in the parameter tensors, which are often in a low rank. The tensor can be regarded as sparse when the zero elements dominate the values, otherwise the tensor is dense. As the sparsity of the tensor is directly related to the portion of the zero elements, it is natural to use the L_0 -normalization function to figure out the how many zero elements are contained in the model in total. Considering the entire parameter tensor $\mathbb{X}$ is comprised by a series of layer-wise tensors $x_1, x_2, \dots, x_l$, where l is the number of layers. The model sparsity can be calculated as follows:

$$\|\mathbb{X}\|_0 = |x_1|^0 + |x_2|^0 + \dots + |x_l|^0. \quad (5)$$

Under the control of L_0 -normalization, elements with exact value of zero will be marked as 1, otherwise 0. Therefore, one

easy way to reflect the model sparsity can be finally defined as: $\text{sparsity} = \frac{\|\mathbb{X}\|_0}{|\mathbb{W}|}$, where $|\mathbb{W}|$ represents the total number of model parameters.

2) *Pruning Strategy*: Designing a proper pruning strategy requires the consideration of various factors. The most fundamental one is the selection of pruning granularity, *i.e.*, the basic unit for conducting pruning [19]. Generally, we can classify pruning into two categories: (1) fine-grained pruning and (2) coarse-grained pruning. The fine-grained pruning is usually based on the element-wise operations that remove the insignificant neurons by checking the weights and activations in sequence. This method can minimizing the model size while incurring huge computational overhead for searching all the parameters, especially in the case of training deep models. In contrast, the coarse-grained pruning conquers this limitation by inspecting the neuron significance in the group-wise manner, which filters the neurons according to the tensor shape and parameter size. Therefore, this kind of pruning can effectively reduce the searching complexity and is easy to deploy.

Another important factor is how to schedule the pruning in the training procedure. During the runtime of the pruning algorithm, the one-shot pruning [20] is the most straightforward implementation, which just prunes the model once during before training convergence. Different from the one-shot pruning, the iterative pruning filters the significant neurons, removes the trivial ones and reconstructs the network function by fine-tuning the compressed model in a repetitive manner [21]. In this condition, the starting and termination of the pruning process will significantly impact the model training efficiency, which can be jointly optimized with data quantization [24, 78] and gradient sketch [51, 79].

3) *Pruning Metrics*: Generally, there are four crucial issues should be decided when implementing a high-performance pruning algorithm, including (1) precisely measuring the training contribution of all the neurons, (2) correctly determining which neurons can be removed from the model, (3) effectively re-tuning the compressed model after pruning while not degrading the final training quality, and (4) making the pruning operations online applicable for the production scenarios.

As to the first issue of defining each neuron’s training contribution, Han *et al.* [22] employed the regularization remainder term in the loss function of the model to guarantee the network sparsity of the parameter tensors. By defining the significance thresholds in advance, it is readily to reflect each neuron’s contribution that neurons yielding small value of parameters (*e.g.*, weights and activations) will be considered as insignificant, thus can be removed from the model. Although this method is easy to implement, the theoretical assumption for introducing the regularization term may not be hold in practice, which will destroy the integrity of the function blocks in the network and finally misleads the prediction results.

Meanwhile, aiming at designing an efficient pruning algorithm, Dong *et al.* [23] introduced the second-order partial difference based on *Taylor Series* to unfold the original loss function and made the approximation to measure the contribution of each neuron. Although the idea of this method is straightforward, the calculation of the second-order Hessian matrices will bring huge computation overhead, which is not

Fig. 4: The workflow of knowledge distillation.

feasible for realistic implementation. One potential trade-off to reduce the computational pressure is to restrict the gradient approximation to a portion of layers instead of the entire network model. Bounded by the linear convergence speed related to the approximation error in each layer, this second-order based method can effectively reduce the model size while not degrading the model training quality and inference accuracy.

Furthermore, as to handling the subsequent model reconstruction and fine-grained re-tuning, Han *et al.* [24] proposed a comprehensive pipelining-aware pruning method based on three key stages: (1) removing the redundant neurons with less significance, (2) quantizing the weights and activation of the compressed model and (3) using Huffman coding to simplify the data representation. These three steps are jointly considered to reduce the memory footprint during learning runtime and providing a good training accuracy.

Finally, in order to conduct online pruning, Guo *et al.* [25] achieved this goal by combining the parameter pruning with network training and dynamically adjusting the model structure. For implementation, the loss function is optimized by adding the sparsity-based regularization and parameters near zero are removed from the forward propagation process, so as to reduce the model size and yield less gradient calculation during backward propagation stage.

4) *Summary*: Overall, the network-aware parameter pruning methods have shown promising advantages to compress model size, reduce computational overhead and save memory footprint. Therefore, we can employ parameter pruning to accelerate the on-device learning applications and guide the design of the corresponding frameworks.

B. Knowledge Distillation

Different from the model compression technologies that significantly change the structure of the original neural networks, knowledge distillation is also a promising method to obtain a small model that provides same function of the original big model, without the manual adjustment of loss function and update rule.

Generally, the gist of knowledge distillation is to train a small model called *student* to imitate the behaviour and function of a large model called *teacher*. This training dependency can be marked as the *teacher-student* scheme. In realistic implementation, the student model is often shallow and simple, comprised of common neural blocks (*e.g.*, FC and CONV layers). For example, a simple five-layer based CNN network can effectively approximate the behaviour of the ResNet18 model [80]. Specifically, Breiman *et al.* [26] pioneered the idea

that employed a simple model to provide similar functions of multiple tree-based models. This idea was leveraged by Bucila *et al.* [27] to propose the knowledge distillation method for neural network models. Besides, Hinton *et al.* [28] extended this method to more general cases. As most of the existing knowledge distillation implementations are based on the work of Hinton [28], we will present the rationale of the general distillation.

The name of knowledge distillation indicates that the student model learns the knowledge of how to minimize the loss function of the forward propagation stage by utilizing the prediction probabilities from the teacher model. As to the model training for image classification, a pertinent ML application, the knowledge obtained from the teacher can be reflected by the classification distribution of different labels, which is closely related to the softmax block at the end of neural network. Therefore, the student model can introduce the loss function from the teacher to revise the loss function calculated by its own. This is a natural idea to help the student quickly obtain the prediction skills from the “sophisticated” teacher and can effectively accelerate the training convergence speed, over the learning from scratch. This is a little similar to the gist of *Transfer Learning* [81]. However, the key difference is that knowledge distillation requires a pre-defined student model and needs to modify its loss function.

1) *Combination of Loss Functions*: Although the basic idea is to extract knowledge from the teacher model, we cannot simply employ its vanilla loss function. As the teacher model is well trained with high prediction accuracy, the output class corresponding to the ground-truth label dominates in the classification distribution, while other classes are approximately zero after the softmax block. In this case, even employing the teacher’s loss function cannot bring extra useful information to the student, over the vanilla loss function based on the ground-truth labels. An effective method to solve this problem is to employ the temperature-based softmax function [82] to adjust the original logits from FC layers and balance the unscaled log probabilities of shape. The temperature-based softmax can be described as:

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}, \quad (6)$$

where p_i is the classification probability of each class i and z is corresponding logits inside the softmax function. Besides, T is the hyper-parameter called temperature to control the probability distribution of the output. Note that we will get the standard softmax function when T is 1 and the probability distribution will become softer when setting a higher value of T , so as to make the teacher model provide more effective information for the student model’s learning. The key that a teacher can successfully transfer the knowledge to the student is to get the prediction experience from the teacher’s loss function, which is called the *dark knowledge* [83]. The entire distillation procedure is actually based on the transferring of this dark knowledge, from the teacher to the student. Therefore, the core modification of the student’s loss function (*i.e.*, overall loss) can be formulated as:

$$\mathcal{L}\{x; w\} = \alpha \cdot \mathcal{L}_s\{y, \sigma(z_s; T = 1)\}$$

$$+ \beta \cdot \mathcal{L}_d\{\sigma(z_s; T = t), \sigma(z_d; T = t)\}, \quad (7)$$

where x , y , w and σ represent the input data, ground-truth label, model parameters and the softmax block, respectively. Besides, z_s and z_d are the logits of the student and teacher, respectively. Note that the overall loss function $\mathcal{L}$ contains two parts: the student loss $\mathcal{L}_s$ and distillation loss the $\mathcal{L}_d$, under the control of the coefficient hyper-parameters α and β . We use Fig. 4 to better illustrate the workflow of knowledge distillation with these two kinds of loss functions. The student loss captures the distance between predicted class and the ground-truth labels marked in the dataset, under the configuration of $T = 1$. Meanwhile, the distillation loss measures the difference between student's loss and the teacher's loss, which can be used to reflect the dark knowledge learned from the teacher and the temperature inside is $T = t$.

2) *Tuning of Hyper-parameters*: It is worth noting that the configuration of the hyper-parameters of α , β and t is an error-and-trial procedure, which significantly impacts the distillation efficiency. In the implementation of existing methods [84, 85], the temperature value t is often set as $t \in [1, 20]$. Empirically, a lower temperature makes sense when the student model is much smaller over the size of teacher model. The core reason of this setting is that a higher value of the temperature will bring too much information from the distillation loss, which may easily exceed the learning capacity of the student model [86]. However, as the true learning capacity of a student model is hard to measure in prior, the setting of temperature t still requires future research to obtain a more intelligent and efficient controlling. Meanwhile, the values of α and β also require careful setting because these hyper-parameters reflect the learning weight between the student loss from the ground-truth labels and the distillation loss from the teacher's experience. A general setting is to make $\alpha + \beta = 1$ and α is often smaller than β [28]. Some recent researchers also focus on how to adjust these hyper-parameters in a more flexible manner [87].

3) *Usage of Model Training*: On top of the tuning of loss function and coefficient hyper-parameters, we can easily generate a light-weight student model that holds similar performance of the complex teacher model. More precisely, at the beginning of each iteration, the data samples will input to both student and teacher models. After the forwarding propagation to calculate the output prediction, the student model can adjust its loss function by introducing the teacher's loss. This pattern ensures that the student can learn from ground-truth labels and teacher's experience. In the backward propagation, the student model will calculate the gradients based on the adjusted loss and use these gradients to update the model parameters. Therefore, the much simpler student model can provide comparable prediction accuracy as the teacher. Considering the computational demands of conducting the knowledge distillation procedure, the student model is often pre-trained on servers for model convergence. After the student model is trained, we can deploy it on embedded platforms to handle the on-device learning applications.

4) *Summary and Extension*: Introducing knowledge distillation into other model compression technologies (e.g.,

Fig. 5: The workflow of model fine-tuning.

network-aware parameter pruning in §III-A and low-precision data representation in §IV-B) can further promote the efficiency of neural network training. For example, Polino *et al.* [29] trained a quantized student model from the full-precision teacher in the forward propagation stage. Theis *et al.* [30] jointly considered the distiller optimization with network pruning. Tann *et al.* [31] proposed a hardware and software co-design for low-precision neural networks by employing the knowledge distillation.

Overall, applying the knowledge distillation approaches to on-device learning can conquer the challenge of huge memory footprint and storage requirement by extracting a small student model with similar performance. This can effectively reduce the inference latency for on-device learning applications.

C. Model Fine-tuning

Recall that the on-device learning applications are often deployed in the resource-constrained environment, where the device's CPU is not permitted for running at top speed for a long time and the data available for training are relatively few. Besides, considering the limited memory capacity on the device, not all the neurons or layers can participant in the training procedure. Therefore, handling the on-device learning requires the careful design of model structure and computational efficiency.

A promising way to achieve this target is to employ the transfer learning based fine-tuning to update the model. As shown in Fig. 5, the fine-tuning procedure contains two key steps: (1) pre-training a model on large dataset (e.g., ImageNet [88]) and apply the model to the on-device dataset by using transfer learning, and (2) defining a part of the layers as updatable (others are freezed) and fine-tuning these layer during the training procedure. Therefore, the transfer learning and partial-layer fine-tuning are the most fundamental concepts to implement the on-device learning. We will discuss these methods in the following.

1) *Transfer Learning*: Although the conventional ML applications (e.g., image classification [89] and object detection [90]) have been widely deployed in commodity large-scale clusters, their success implicitly relies on the huge-scale of the available training data, with the ground-truth labels marked in advance. However, this assumption cannot be satisfied in the on-device learning environment as we mentioned above [91]. Fortunately, the transfer learning is a promising method to conquer this challenge by using the prediction experience

trained from another dataset, which is usually much larger in data scale. For the expression convenience, we denote the training procedure by using other large-scale dataset as the source task and corresponding training data is marked as the source dataset. The power of transfer learning is to eliminate the limitation of insufficient on-device data by pre-training the model with the source dataset [81], so as to extract more knowledge that cannot be obtained as usual.

Currently, the transfer learning based approaches have been successfully used in many visual computing scenarios, raising the efforts of designing more intelligent and adaptive algorithms [92–94], where the fine-tuning is one of the most promising methods for meeting the requirements of on-device learning acceleration [11]. Specifically, the ImageNet [88] is a suitable source dataset to conduct the pre-training of the model, from which we can obtain rich knowledge about the feature extraction and layout location used in the on-device learning applications. The gist for accelerating the training speed is to reduce the completion time of updating the model parameters in each iteration. As a result, there are two kinds of fine-tuning methods in general: (1) only updating a part of the layers of the network to reduce the computational overhead of matrix operation [32] and (2) updating all the parameters while giving the backbone blocks in the pre-trained model more significance to better extract the classification features of the training data [33]. However, the hyper-parameters of how many layers are updatable and how to design the backbone network are still dependent on the manual configuration, which is often a trial-and-error process. A high-performance on-device learning framework should automatically make the proper decision to control these hyper-parameters. Here, we will present two directions, *i.e.*, (1) layer-wise freezing and updating and (2) model-wise feature sharing to achieve this target in the following.

2) *Layer-wise Freezing and Updating*: The layer-wise freezing and updating can control the fine-tuning of each neuron's parameter. Generally, the learning process requires a pre-trained model that owns the knowledge of feature extraction from a large-scale source dataset. Then, the learning process will continue by re-training the model with the on-device local dataset. Different from the traditional ML applications that usually train from scratch with the random initial parameters, the transfer learning based on fine-tuning can provide the similar training quality while requiring much fewer iterations until convergence [95, 96].

As to the implementation of fine-tuning, we need to define the freezing layers and the updatable layers first. The parameters of the freezing layers are fixed as constants that will not participant in the training process. In contrast, the updatable layers are the truly active part of the neural network, where the parameters keep changing until training convergence. Briefly, we can regard that the freezing layers store the knowledge learned from the large-scale source dataset and the updatable layers generate the customized models by using the local on-device dataset. The strategy of updating the partial layers instead of the entire model comes from the consideration that the training based on the small-scale of on-device dataset will easily lead to the model over-fitting [97], even with the support

of pre-training with source dataset. In order to maintain the knowledge of the pre-trained model, only the last few layers can be updated, while other layers should be freezed to persist the low-level extracted features [34]. Note that the hyper-parameters of how many layers should be freezed still relies on the empirical configuration, which may restrict the flexibility of on-device learning when facing with new networks.

However, considering the prerequisites of high-accuracy pre-trained model and trial-and-error adjusting of updatable layer hyper-parameter, some compact models tailored for the mobile environment, *e.g.*, the ShuffleNet [98] and MobileNet [99] may not well compatible with the fine-tuning method because their relatively shallow network structures do not have enough capacity to express the full parametric information of original deep models. Researchers need to balance the trade-off between runtime model size and overall training quality, and design general methods to handle the training configurations automatically.

3) *Model-wise Feature Sharing*: Apart from the fine-tuning based on layer-wise controlling, we can also apply the knowledge of the entire pre-trained model to the on-device learning. As different on-device tasks may own the similar learning objective, it is natural to share the features across models by transferring the corresponding parameters. Some researchers have paid efforts to share the knowledge of feature extraction between different models [35]. Actually, these existing methods are mainly designed for the image classification applications, of which the model is usually shallow with relative less knowledge [36]. However, in order to capture richer learning patterns, the models have become much deeper and bring new challenges to on-device learning. The key difficulty is how to transfer model knowledge from deep models and make full use of this knowledge when the computational and memory capacity is limited. Therefore, the pertinent learning paradigm, *e.g.*, the vanilla life-long learning [100], is not suitable for the on-device learning environment. However, it is possible to extend this method by combining the model compression technologies.

4) *Summary*: The transfer learning based fine-tuning is a promising method to conquer the challenges of on-device learning, including few training data, limited computational resources and strict parameter update rules. In brief, this method can providing the following four advantages for on-device learning: (1) reducing the computational overhead of network training, (2) avoiding model over-fitting caused by few training data, (3) accelerating the entire training speed, and (4) providing highly personalized models for various applications.

IV. TRAINING ALGORITHM OPTIMIZATION

On top of the design of high-efficiency neural networks, the next step is to deploy these models in the edge environment, where computational resources are often scarce and expensive. Therefore, it is necessary to conduct the optimization of training algorithms to reduce the runtime overhead. This target can be achieved by the following two research topics: (1) data-adaptive regularization and (2) low-precision data

representation. The gist of these two topics is to alleviate the computational overhead of matrix operations, so as to improve the on-device hardware efficiency.

A. Data-adaptive Regularization

In order to conquer the challenge that the available training data of on-device learning are usually in small scale, it comes the development of data-adaptive regularization, which make the optimization function match the user personal data while avoiding model overfitting. We can achieve this target by modifying the loss function and learning objective of the original model, *e.g.*, adding $L1$ - or $L2$ -normalization to the loss function. This kind of lightweight modification on algorithm optimization shows promising advantages to the acceleration of on-device learning, especially for the resource-constraint edge environment.

1) *Core Formulation*: As regularization focusing on producing a smaller model with the approximate performance of the original one by adjusting the loss function of neural network, the statistic efficiency of model generalization should be bounded in an acceptable range, where controlling the generalization error need to be consider in the first priority. Here, we will present the basic formulation of regularization first. Generally, guaranteeing the generalization of small models often resorts to the restriction of the corresponding model capacity of searching the optimum. A typical way is to reduce the variance caused by the bootstrapping [67] in the *Stochastic Gradient Descent* (SGD) algorithm and its variants [68, 101]. Therefore, the general formulation can be described as:

$$\mathcal{L}(w; x; y) = \mathcal{L}_0(w; x; y) + \lambda \cdot R(w), \quad (8)$$

where $\mathcal{L}$ and $\mathcal{L}_0$ represent the overall training loss and the error of the objective function, respectively. Note that the $R(w)$ represents the regularization function to adjust the original loss, which is usually based on the $L1$ - or $L2$ -normalization. Besides, $R(w)$ is under the control of the hyper-parameter λ called *regularization strength* [102] that makes the balance between training loss and regularization error.

2) *On-device Network Sparsification*: The power of regularization is closely related to the optimization method based on neural network sparsification. In previous research, Han *et al.* [37] proposed the *Dense-Sparse-Dense* (DSD) method to improve the prediction accuracy after regularization by pruning the original model. As a pertinent method to implement regularization, the key of sparsity is to relax the optimization constraints of the model and make the search direction get rid of the saddle point more easily, especially when the network fall into a local minimum, so as to finally avoid model over-fitting and provide robust generalization. Consequently, there are two kinds of widely-used methods to design the regularization function: (1) $L1$ -normalization and (2) $L2$ -normalization.

In the $L1$ -normalization based method, the regularization is enabled by utilizing the element-wise sparsity inside the

network. We can add the following remainder term to the loss function:

$$R_1(w) = \sum_{i=1}^{|W|} |w_i|, \quad (9)$$

where $|W|$ is the total number of model parameters.

Meanwhile, as to the $L2$ -normalization, the remainder term can be represented as follows:

$$R_2(w) = \sum_{l=1}^L \sum_{i=1}^n |w_{l,i}|^2, \quad (10)$$

where L and n represent the number of layers and per-layer weights, respectively. Note that these two remainder terms can be jointly used and the loss function is modified as:

$$\mathcal{L}(w; x; y) = \mathcal{L}_0(w; x; y) + \lambda_1 \cdot R_1(w) + \lambda_2 \cdot R_2(w). \quad (11)$$

The main difference between these two kinds of regularization is that $L1$ -normalization simplifies the model and restricts the model training capacity by making a portion of the parameters as zero. However, this kind of simplification may lead to model over-fitting and degrades the model training quality. Observing these limitations, the $L2$ -normalization does not require the parameters to be zero while still providing a good training accuracy, compared with the original model.

3) *Block-wise Regularization*: Although introducing regularity can simplify the computational operation of model training, conducting element-wise sparsity may still require fine-adjustment of each parameter, which is hard to implement in practice. Recently, some researchers have focused on the block-wise regularization that control a group of parameters, instead of the single element. Specifically, a portion of group elements are set as zero together to conduct sparsification.

The key idea of block-wise regularization is to add penalty to the data loss following the perspective of all the groups in each layer. Therefore, by defining the group parameters in layer l as w_l^g , the basic formulation can be described as:

$$\mathcal{L}(w; x; y) = \mathcal{L}_0(w; x; y) + \lambda_r \cdot R_r(w) + \lambda_g \cdot \sum_{l=1}^L R_g(w_l^g), \quad (12)$$

where λ_r and λ_g represent the conventional regularity strength and the group-wise strength, respectively. Moreover, the group-wise remainder term can be further defined as:

$$R_g(w_l^g) = \sum_{g=1}^G \|w_l^g\| \quad (13)$$

$$= \sum_{g=1}^G \sum_{i=1}^{|w_l^g|} (w_l^g)^2, w_l^g \in w^l, \quad (14)$$

where $|w_l^g|$ represents the element numbers of the parameter group w_l^g . Note that we can use the *Group Lasso Regularization* [38] in the form of $\lambda_g \sum_{l=1}^L R_g(w_l^g)$ to accumulate all the magnitudes of the parameters in the group manner.

INT8-based Operation	Energy Saving over FP32	Memory Reduction over FP32	Processing Speedup over FP32
Addition	30.16×	115.33×	3.32×
Multiplication	18.53×	24.98×	7.45×

TABLE III: Efficiency comparison between INT8 and FP32-based operations.

4) *Summary*: Block-wise regularization belongs to the category of fine-grained sparsity that explores the value pattern and matrix shape of the parameters. In recent researches, the group partition metrics can be defined in various aspects, including *Convolutional Neural Network* (CNN) kernel numbers, output feature maps, filter sizes and network layers. For example, Mao *et al.* [39] balanced the regularity of sparse structure and inference accuracy by designing a coarse-grained pruning based on CNN characteristics. Besides, Anwar *et al.* [40] proposed the structured sparsity by exploiting the pattern of intra-kernel strides to reduce the model complexity, including kernel size and output feature maps. All of these approaches can effectively reduce the computational pressure of the learning process while not degrading the final prediction accuracy and model quality. Overall, the network sparsification employed inside data-adaptive regularization can simplify the network structure and reduce the inference latency, which is a fundamental evaluation metric for on-device learning acceleration.

B. Low-precision Data Representation

Quantization aims to reduce the bit numbers of data value, *i.e.*, representing a number in low precision. In the on-device learning applications, the data type used in arithmetic operations and matrix calculations usually follows the floating-point value with 32 bit width, called the FP32 format. However, in the resource-constrained on-device environment (*e.g.*, limited I/O bandwidth and scarce computational primitives), the conventional FP32-based operations will easily exceed the available computational capacity, which is impractical for swift model training and low-latency inference. Therefore, it is meaningful to reduce the computational overhead by optimizing the data representation. Fortunately, previous researches have shown that the neural networks are often over-parameterized, where the model parameters are in high redundancy [103]. This phenomenon indicates that representing the data in the fully FP32 precision is not necessary and is possible to handle the parameters in lower precision while not degrading the model quality. A common way is to transfer the FP32 data type into the fixed-point value with 8 bit width, called the INT8 format. Actually, it is feasible to use even lower bit width for data representation, including the binary [41], ternary [42] and XNOR [43] quantization.

Considering the iterative pattern of model training with forward and backward propagation, the weights, activations and gradients can be quantized together for better system efficiency, including reducing per-iteration time, decreasing memory footprint and saving energy cost. More precisely, as shown in Table III, we compare the efficiency between INT8 and FP32-based operations in the above metrics, so

Fig. 6: Common value mapping schemes used in data quantization.

as to highlight the feasibility of reducing resource cost by employing INT8-based quantization. As a result, quantization is a hot topic in the field of on-device learning acceleration. In the following, we will discuss quantization in three aspects: (1) the preliminary knowledge about quantization algorithms, (2) the post-training quantization for on-device inference and (3) the quantization-aware training for on-device online learning.

1) *Preliminary Knowledge*: In order to better understand the rational of quantization, we will introduce five key concepts used in quantization algorithms: (1) value mapping, (2) zero point, (3) linear scaling, (4) range clipping and (5) integer rounding.

Value Mapping. The gist of quantization is to convert the data value from the original fully precision FP32 format into the lower precision format (*e.g.*, INT8). For expression convenience, the following discussion is based on the INT8-based quantization. In a macro view, quantization can be regarded as a mapping function that transfers the wide-range floating-point number to a small-range fixed-point value, covered by 8 bit width. This primary transformation of quantization can be formulated as:

$$q = \frac{r}{s} + z, \quad (15)$$

where q and r represent the quantized value and the original number, respectively. Besides, the scale s is the core of the mapping function that controls the final range of the quantized value. Also, the zero point z (also called the quantization bias) is an optional parameter to re-locate the base point of the number axis by shifting the quantized range with an offset. Meanwhile, as the tensors are proceed across the layers in sequence, the quantized value in a layer may need to recover to the original FP32 range before entering the next layer. This procedure is called the dequantization, which can be described as:

$$r = (q - z) \cdot s. \quad (16)$$

Zero Point. As the zero point is used to adjust the quantized range with an offset, the value mapping algorithms can be classified into two categories: the symmetric scheme (in Fig. 6(a)) and the asymmetric scheme (in Fig. 6(b)). The symmetric scheme can directly map the original range according to the absolute value between the maximum and minimum. Thus, it does not need the involvement of the zero point and is easy to implement. In contrast, the asymmetric scheme determines the zero point based on the scale range to fully utilize the quantization bit width.

Fig. 7: Range clipping removes the outliers according to the data distribution.

Linear Scaling. We can observe that the core of quantization is the division operation based on the scale s . Therefore, we need to determine a suitable scale because simply dividing the original number by a constant scale will lead to the mismatching of data distribution between original exponential interval of FP32 and the uniform interval of INT8. Besides, as the elements of the parameter tensor often locate in a sharp but wide domain, it is necessary to make the quantized range cover the original data range. A common way is to consider the difference between the maximum and minimum, with the constraints of limited bit width. Therefore, the formulation of scale s can be described as:

$$s = \text{scale}(a, b, n) = \frac{b - a}{2^n - 1}, a < b, \quad (17)$$

where a and b represent the starting and end point of the quantized range, respectively. Besides, n is the limited bit width, e.g., $n = 8$ for INT8 asymmetric quantization.

Range Clipping. Considering the wide-range of the original FP32 values, it may be impossible to cover the entire range via limited bit width. In this condition, we need to remove the outliers in both ends of the data distribution. As shown in Fig. 7, this procedure is called the range clipping, which can be described as:

$$\text{clip}(r; a, b) = \min\{\max\{r, a\}, b\}. \quad (18)$$

Integer Rounding. In order to match the integer type of INT8 quantization, the results should be rounded after conducting the division operation. The *Stochastic Rounding* [104] is one of the most widely-used methods, which can be described as:

$$\text{round}(r) = \begin{cases} \lfloor x \rfloor, & w.p. \quad 1 - \frac{x - \lfloor x \rfloor}{\delta} \\ \lfloor x \rfloor + \delta, & w.p. \quad \frac{x - \lfloor x \rfloor}{\delta} \end{cases}, \quad (19)$$

where δ represents the smallest interval of the fixed-point numbers with the given bit width. Combining the integer rounding, the common formulation of quantization can be finally described as:

$$q = \text{round}\left(\frac{\text{clip}(r; a, b)}{\text{scale}(a, b, n)} + z\right). \quad (20)$$

On top of these five key concepts, the quantization algorithms for on-device learning can be designed, mainly for two application scenarios: (1) *Post-training Quantization* (PTQ) for inference and (2) *Quantization-aware Training* (QAT) for online learning.

2) *Post-training Quantization*: In inference, the network models can be directly quantized without further re-training. This kind of methods is called the *Post-training Quantization* (PTQ) [44]. Due to the static property of model parameters, PTQ often collaborates with the fine-tuning approaches mentioned in §III-C, where the scale parameter is core factor to dynamically adjust the quantized range and restrict the tensor values in the integer data type. As the scale parameter depends on all the tensors in each layer, we need to capture both maximum and minimum in the tensor, and transfer the numerical value from FP32 to INT8 format. This kind of single scale parameter should be independently calculated in the layer or channel-wise granularity to capture the value range properties, especially when the data distributions vary greatly across different layers and channels.

After correctly determining the scale parameter, the weights and bias can be easily controlled by quantized them just once at the end of model training. Besides, for better model quality, the activations can be quantized according to the data distribution of the original FP32 tensors. A common way is to collect the complete statistics of the activations before model deployment, i.e., following the offline scheme. In practice, the on-device learning framework can get these key information by executing a few trial calibrations on the original FP32-based model. Thus, the corresponding scale parameter can be fixed without further changing. However, as the offline scheme cannot observe all the potential data distributions in advance, the outside values may need to be clipped during the inference runtime, causing the degradation of the prediction accuracy. Therefore, we can employ the online scheme to eliminate the limitation from the range clipping by measuring the runtime maximum and minimum of the tensors. It is also worth noting that this dynamic inspection will bring extra computational operations that aggravates the on-device learning overhead.

However, some recent researches have shown that the PTQ-based methods may not effectively guarantee the model accuracy under the INT8 data format [105, 106]. Besides, some dedicated models, e.g., the MobileNet [99] and ShuffleNet [98] cannot adapt to the assumptions of PTQ because the small representation capacity of the INT8 model may not capture all the learning knowledge stored in the original FP32 version.

3) *Quantization-aware Training*: Recall that quantizing a FP32-based model into the INT8 version without further re-training may lead to the degradation of model quality and prediction accuracy, which is hard to meet the requirements of on-device learning. As a result, it comes the *Quantization-aware Training* (QAT) [45] to address the above limitations.

Different from the fixed PTQ-based approaches, the QAT algorithms introduce the quantization of both forward and backward propagation stages into the entire training procedure, where the data representation of weights, activations and gradients are simplified from original FP32 format into low bit width. By employing QAT, the system-level performance bottleneck caused by the limited I/O bandwidth, little memory capacity and scarce computational primitives can be well alleviated, so as to fundamentally accelerate the deployment of on-device learning. Here, we use Fig. 8 to illustrate the detailed workflow of QAT in a given layer, including the FP

Fig. 8: The workflow of QAT of FP and BP stages.

and BP stages.

FP Stage. In the forward propagation, both the weights and activations are quantized [46]. More precisely, the input data and weights are quantized from FP32 into INT8 format before entering the operations of matrix calculation, including the dot product of FC layers and the convolution multiplication of CONV layers. After the optional batch normalization to calibrate the data distribution of intermediate values, the results from the activation function will also be quantized and finally becomes the output of this layer [47]. Besides, the quantized activations may need the range clipping to better match the data distribution [48]. Note that the INT8-based output will be dequantized as FP32 format by multiplying the scale factor used in the quantization, before serving as the input into the next layer. In the next layer, the FP32-based input will be quantized again, divided by the independent scale factor based on the value distribution of the next layer. Therefore, each layer has its own scale factor and this quantization/dequantization procedure will be conducted in sequence from the first layer to the last layer in the FP stage.

BP Stage. In contrast, as to the quantization of backward propagation, the partial differential for calculating the gradients will be conducted from the last layer to the first layer, following the chain rule [55]. The gradients of the last layer will be quantized before the differential operation, where both activation and weights are represented in the INT8 format. Therefore, the gradients of activation and weights of current layer are also calculated as INT8. After obtaining the full gradients of this layer, we need to dequantize them into FP32 format and update the model parameters in full precision for better optimization accuracy [49]. Therefore, a full-precision copy of the original weights and activations is needed for conducting the model updating in the FP32 data type.

Gradient Sketch. It is worth noting that quantizing gradients can also reduce the communication traffic that needs to be exchanged among different workers, if we deploy the on-device learning in a distributed manner. One of the most common ways is to use the sketch-based methods [50, 51, 79] to quantize gradients and accelerate the flow transmission process, including the frequency sketch [107] and quantile sketch [108]. The gist of the sketch methods is to approximate the tensors by mapping the elements into several discrete values. The key difference between them is that the former

uses the frequency of element occurrence while the latter is based on the estimation of the value distribution in the tensor. As the quantized parameters are mapping from the continuous distribution to the integer-based discrete variables, it is hard to obtain the optimum by using the conventional gradient-based methods due to the large approximation error in partial differential operations [103]. Fortunately, we can use the finite-difference estimators or continuous relaxation estimators to address this problem [109].

Overall, both weights and gradients are compressed before conducting the dot production between tensors. The arithmetic operations are entirely based on the INT8 data type, so as to save the processing completion time. After the dot production is done, the tensors of weights and gradients will be dequantized to compensate for the accuracy degradation caused by the rounding operation. The quantization procedure of FP and BP stages will be iteratively conducted until the model convergence [45]. After the model is trained, the low-precision weights and activations quantized in the last iteration will be used for the inference purpose, serving as the essential function of the FP stage.

4) *Summary:* In the confrontation with the performance bottleneck caused by the limited I/O bandwidth, low memory capacity, scarce computational primitives and network transmission latency of on-device learning in practice, data quantization is a promising method to address these challenges by representing the data value in a relatively low precision while not hampering the final quality of model training. As to the quantization methods, the PTQ scheme aims at compressing the model size after training is completed, which is often used in the offline inference scenarios and does not require large amount of data. However, the QAT scheme trains a quantized model from scratch and holds higher accuracy, which often collaborates with the fine-tuning and transfer learning techniques. We believe that data quantization can be combined with other optimization methods, *e.g.*, network pruning and knowledge distillation to further reduce the computational overhead and accelerate on-device learning applications.

V. HARDWARE IMPLEMENTATION

Apart from the model-level neural network design and algorithm-level training optimization to implement modern ML applications, deploying the on-device learning tasks and

accelerating corresponding processing speed also need the support of specific hardware, so as to conquer the severe challenges from the resource-constrained environment. In order to guide the readers to achieve this goal, we will discuss the hardware-level implementation of on-device learning in four aspects: (1) embedded memory controlling, (2) dedicated computational primitives, (3) low-level instructions, and (4) MIMO-based communication. Considering these four optimization factors, the hardware implementation can fundamentally accelerate the processing speed of on-device learning applications.

A. Embedded Memory Controlling

As modern learning applications often rely on complex neural networks requiring huge memory footprint, addressing the challenge of limited available memory is one of the most crucial issues for on-device learning deployment. During the runtime of on-device learning, the memory is often used for the storage of intermediate results and the loading of model parameters. As the data access speed in memory is much slower than that in the CPU cache or registers, the cost of in-memory access is expensive for the on-device applications. Besides, edge devices usually equip with quite limited memory, which is not sufficient enough to handle modern ML applications. For example, as to the image classification based on the relatively shallow ResNet18 [80] model (in Table II), the storage of network and parameters even yields megabytes of memory footprint.

Therefore, the on-device learning frameworks should carefully schedule the task to avoid the oversize of memory space [11]. Venkataramani *et al.*, [52] proposed the SCALEDEEP system that jointly exploits the communication and computational properties of model training to better optimize the memory assignment and inter-connection between different machines. SCALEDEEP leverages the prior information of floating-point operation per second (FLOP) and the FLOP ratio in different layers of the neural network, so as to optimize the memory management in a hierarchical architecture and minimize the overhead of parameter synchronization. The efficiency of embedded memory management can be further improved by employing the method of proper data movement and fine-grained pipelining [53, 54] that overlaps the computation and communication [55, 56].

B. Dedicated Computational Primitives

In addition to the memory management, dealing with the huge computational overhead from the matrix operation is also a significant direction for on-device learning acceleration, which requires the optimization of computational capacity. However, the software-level optimization cannot bring fundamental improvement of reducing computational pressure and accelerating the processing speed. In order to meet the demands in the production environment, a hot research direction is to design dedicated computational primitives and the corresponding domain-specific instructions to drive this hardware on device.

Actually, the dedicated hardware have been widely studied for the acceleration of distributed processing. Chen *et al* proposed the multi-chip based system, called DaDianNao [57], to accelerate general ML applications. This system jointly considers the algorithmic characteristics of neural networks and communication patterns of internal connection based on Fat-Tree [110] topology, so as to provide a flexible and customized neural network engine for large-scale ML training applications. Although these ML systems stem from the cloud environment, their idea of using dedicated hardware to simplify the software calculation and accelerate the learning speed can be adopted to the on-device learning scenarios. A pertinent industrial case is the neural engine in Apple CPUs [58] designed for the Animoji [111] and augmented reality applications on iPhone. Besides, the Huawei NPU in Kirin CPUs [59] also facilitate the intelligent photograph rendering for portraits and night scenes. Moreover, Settle *et al.*, proposed the Xilinx [60] neural network engines and design the low-precision data representation protocol, which implements the 16-bit floating-point and 8-bit fixed-point quantization algorithms into the FPGA-based hardware. This kind of dedicated neural network chips can provide significant training speedup over the commodity GPU-based machines.

C. Low-level Instructions

On top of the dedicated computational primitives, fully exploiting the hardware power requires the corresponding support of underlying low-level instructions, which are employed to drive the hardware on edge devices. These instructions and the corresponding primitives are often called as the domain-specific hardware [61].

Recently, some researches have focused on this direction. Wu *et al.* [62] built Phoenix, a quantization-aware processor for low-precision FP32 operations to address the challenges of hardware low efficiency and accuracy degradation when quantizing the floating-point numbers. The core instruction is to design a specific floating-point multiplier with 8 bit width to reduce the memory and computational overhead. These instructions are embedded in the FPGA-based processors and show better performance over the INT8 quantization. Besides, McDanel *et al.* [63] proposed a hardware acceleration system that aims at improving the inference speed of learning applications by implementing the full-stack optimization methods on FPGA-based chips, including the neural network models, computational paradigm, and chip design. The system maintains good balance between model accuracy and hardware efficiency. Moreover, it is possible to replace the time-consuming multiplication and division operations by the low-level left/right shifting instructions, which are easy to implement on commodity CPUs and can significantly save the computational cost. Therefore, leveraging the power of dedicated hardware and its underlying instructions is a promising method to accelerate the on-device learning applications.

D. MIMO-based Communication

As to extending the application scenario of on-device learning, it is possible to handle this procedure in a distributed

manner via mobile communication. In this condition, the communication overhead will also become the bottleneck of system performance. Fortunately, we can employ the *Multiple-input Multiple-output* (MIMO) techniques to reduce communication cost of on-device learning. Generally, a MIMO-based device often equips with massive antenna arrays. By exploiting the superposition nature of radio signals, the *over-the-air computation* enables a group of devices to simultaneously transmit their model updates, and the server directly recovers the summation of these updates. The corresponding encoding and decoding algorithms essentially promote the capability of wireless infrastructures and accelerate the on-device training process.

In practice, the MIMO-based systems can be widely used for many mobile scenarios. For example, Vu *et al.* [64] optimized the cell-free MIMO transmission to support federated learning over mobile devices, which can be regarded as a distributed case of on-device learning. By capturing the inherent interactions and transmission characteristics of the training participants, the efficiency of energy saving and processing speed can be jointly improved, which are two critical factors to on-device learning. Besides, in order to exploit the physical-layer property, T. Huang *et al.* [65] designed the *Physical-Layer Arithmetic* (PhyArith) that applies coding techniques in mobile communication. Considering the on-device learning demands, a mobile device can encode its gradients and exchange them with other partners via the inherent superposition of radio frequency (RF) signals, when the learning task needs the distributed collaboration. Note that this physical-layer hardware compatibility can be jointly optimized with the algorithm-level compression methods (*e.g.*, data quantization) mentioned in this survey.

Overall, the MIMO-based communication is suitable to the over-the-air environment of the edge devices and can effectively accelerate the learning speed.

E. Summary

Fully exploiting the power of neural network design and training algorithm optimization requires the support of suitable underlying hardware. We stand from four aspects to show how to connect the real-world on-device learning demands to hardware implementation. This discussion could serve as a guide for researchers to build the learning frameworks.

VI. DISCUSSION AND FUTURE DIRECTIONS

Following the guidance of the afore-mentioned three research topics, we present the insights of applying edge intelligence in real-world scenarios. The software and hardware synergy of all the approaches provide implementation details to address the challenges when building high-performance learning systems. On top of the discussion on these on-device learning techniques, we will present the system performance when applying different methods and introduce the potential research topics that could extend the power of on-device learning in future work.

A. Method Evaluation

In the edge intelligence scenarios, the on-device learning techniques are usually implemented on embedded platforms or mobile devices, where object detection and image classification are two pertinent applications. Therefore, we mainly focus on the neural network complexity and system performance of these applications when adopting different methods mentioned in this survey.

As on-device learning applications are usually based on the power of neural networks, restricting their complexity will fundamentally impact system efficiency. Here, we mainly consider the following seven basic metrics: (1) model size, (2) inference latency, (3) training speed, (4) arithmetic operation overhead, (5) memory footprint, (6) model quality, and (7) energy cost. Specifically, as to the image classification application, we also inspect the performance of prediction or test accuracy. Besides, as to the objective detection applications, we focus on the performance of (1) Intersection over Union (IoU) and (2) Mean Average Precision (mAP). All of these metrics are evaluated by applying different methods mentioned in this survey. The overall summaries are highlighted in Table IV. We can observe that the overall system performance can be effectively improved with these methods in different aspects.

B. Intelligent Medical Diagnosis

In recent years, the intelligent medical diagnosis applications based on edge and mobile devices have become a hot research topic. For example, some researchers focus on using one single image of human body for surgery diagnosis [113]. Based on the approaches of pattern extraction, it is possible to analyze the medical images on the user end by utilizing the on-device learning approaches, which can bring great convenience to user's disease screening. Besides, on-device learning methods can automatically diagnose scoliosis on the mobile phone, where sensitive information of user data will not upload to the cloud, so as to well protect the user's privacy. Moreover, it is also a promising direction that employs the on-device learning methods for intelligent tracking and analysis of sports rehabilitation, which can meet the online demands of mobile phones. Consequently, the on-device learning is a promising technique to address the critical issue of privacy leakage and provides real-time processing capacity for medical diagnosis.

C. AI-enhanced Motion Tracking

As the motion tracking applications often hold the high real-time demands, especially in the capture of arm movement, running and other sports, it is natural to employ the on-device learning techniques to enhance the motion tracking quality, including improving the detection accuracy and providing stronger robustness of motion capture. More precisely, using the on-device learning methods can build a more lightweight model on user devices, so as to ensure the real-time capacity and high-level accuracy. For example, some cutting-edge researches aiming at implementing learning-based arm motion tracking for human activity recognition, where smart

TABLE IV: System performance by applying different methods.

Method	System Performance	Related Work
Network-aware Parameter Pruning (§III-A)	Reduce network storage by $12 \times -19 \times$.	[77]
	Restrict the gap of prediction error between original CIFAR-Net within 2.75%.	[23]
	Reduce number of weights by $9 \times$ on AlexNet and $12 \times$ on VGG-16.	[22, 24]
	Reduce number of FLOPs by $3 \times$ on AlexNet and $5 \times$ on VGG-16.	
Knowledge Distillation (§III-B)	Save $9 \times -13 \times$ parameters without drop of prediction accuracy.	[82]
	Restrict the gap of top-1 accuracy between teacher model within 3.4%.	
	Provide $5 \times$ compression ratio and preserve accuracy drop within 0.3%.	
	Preserve mAP of objective detection on PASCAL VOC 2007 dataset by 77.2%.	
Model Fine-tuning (§III-C)	Preserve the error rate on ImageNet within $9.11\% \pm 0.12\%$.	[87]
	Preserve the prediction accuracy of image classification on ResNet-34 with CIFAR10 by $91.7\% \pm 0.09\%$.	[36]
	Preserve mAP of objective detection on PASCAL 2008 dataset by 45.2%.	[35]
Data-adaptive Regularization (§IV-A)	Preserve mAP of objective detection on H3D Human Attributes dataset by 78.98%.	[33]
	Preserve the top-5 accuracy of image classification on DenseNet-121 with ImageNet by 92.21%.	[39]
	Save memory references by up to $2 \times$.	
Low-precision Data Representation (§IV-B)	Improve top-1 accuracy of image classification on VGG-16 by 4.3%.	[37]
	Achieve $4.3 \times$ speedup of training run time.	[50]
	Achieve $7.24 \times$ compression rate of message size.	
	Achieve $1.81 \times$ reduction of per-epoch completion time.	
	Achieve 50% reduction of training and inference run time on MobileNet SSD with COCO dataset.	[106]
	Achieve $2 \times$ latency reduction of face detection.	
	Achieve 76.34% accuracy of image classification on ResNet50 with ImageNet dataset.	[49]
	Achieve 81.9% mAP of object detection on ResNet50 with PASCAL VOC dataset.	
	Achieve $1.63 \times$ speedup in forward stage and $1.94 \times$ in backward stage.	
	Reduce 22% training time on ResNet50 model.	[48]
	Achieve $4 \times$ improvement of peak FLOPs and $4.5 \times$ speedup of inference performance.	
	Preserve light accuracy degradation of 0.07% in IoU and mAP.	[112]
	Achieve $4 \times$ compression ratio of model size.	
	Achieve $1.6 \times$ reduction of energy efficiency on YOLO-3 model.	
	Achieve $1.5 \times$ FPS improvement on YOLO-3 model.	
Embedded Memory Controlling (§V-A)	Reduce 33% memory footprint on YOLO-3 model.	[11]
	Achieve 3.72% higher accuracy and reduce memory footprint on ResNet-50 model.	
	Achieve $1.9 \times$ average frame rate speedup.	
	Reduce energy cost by up to $1.7 \times$ on average.	
	Achieve $7 \times -11 \times$ speedup over vanilla Google TensorFlow.	
	Achieve $4.6 \times -7.3 \times$ speedup of single-precision operations over NVIDIA Pascal architecture.	[52]
Dedicated Computational Primitives (§V-B)	Reduce memory footprint and FLOPs on common CNN models.	
	Achieve 64.4% training speedup with 10Gbps bandwidth in parameter server clusters.	[55]
	Achieve 164.8 $\times$ speedup of CLASS layer over the 64-node GPU cluster.	[57]
	Achieve 300.04 $\times$ speedup of model training over the 64-node GPU cluster.	
	Achieve 150.31 $\times$ reduction of energy cost over the 64-node GPU cluster.	
Low-level Instructions (§V-C)	Achieve 36% storage reduction of model weights and 50% energy reduction of multiplication operations.	[61]
	Achieve 8.14 $\times$ area reduction over the INT8 multiplier.	[62]
	Achieve $7.45 \times$ speedup on VGG-16 model.	
	Improve energy performance of single image inference by up to $151 \times$ over NVIDIA TITAN GPUs.	
	Achieve 30% energy reduction of matrix multiplication.	[63]
	Reduce model size by up to $40 \times$ on AlexNet and $92 \times$ on VGG-16.	
MIMO-based Communication (§V-D)	Achieve 2.28ms inference latency on ImageNet dataset.	[64]
	Achieve 55% reduction of training run time.	
	Achieve $1.5 \times -3 \times$ improvement of communication efficiency when training LeNet-5 model.	[65]

watches are used to precisely track the arm motion in the real-time scenario [114]. With the motion trace of the human poses, the industry can further apply the on-device learning techniques to recognize various body gestures and understand the associated actions (*e.g.*, walking, sleeping, sitting and driving) in human daily life. Compared to the conventional statistical analysis, activities tracking using on-device learning can bring intelligent health monitoring and personalized health analysis to different users. Moreover, the on-device learning methods can also benefit a wide range of social applications, such as natural user interface, motion-based virtual reality, sports and training analytics. In general, on-device learning is a fundamental breakthrough to meet the real-time demands in AI-enhanced motion tracking and can further improve human interactive experience.

D. Domain-specific Acceleration Chips

The on-device learning techniques can be employed in many emerging ML scenarios, where the system performance is often bounded by the limited hardware resources. Therefore, it is a promising research direction that improves the computational capacity by designing the domain-specific AI chips for task acceleration. These chips can be designed from the perspectives of model compression, few-shot learning, quantization-aware training, memory management and the low-level instructions, so as to directly help the developers optimize the training process without too much code modifications. For example, as to accelerating the sparse tensor operations by representing the elements in low precision, the NVIDIA proposed the Ampere architecture [115] to support the quantization-aware training based on the INT8/INT4 format. Besides, these chips can be embedded in the edge devices to provide more powerful computational primitives. A common way is to design the *Application-specific Integrated Circuit* (ASIC) [116] based on

the realistic demands of on-device learning. For example, we can implement the model fine-tuning techniques on mobile phones to faster recognition of human faces. Overall, the research of domain-specific acceleration chips has become a hot open topic and will fundamentally promote the development of learning systems.

VII. CONCLUSION

Driving by the demands of emerging ML applications in the edge intelligence scenarios, it is a trend to transfer the learning procedure from the cloud to the device itself. As to the cutting-edge research of on-device learning, implementing a high-efficiency learning framework and enabling system-level acceleration is one of the most fundamental issues. Therefore, we present a comprehensive discussion on the latest research progress and point out the promising optimization directions, standing in the view of system design. This survey analyses the research frontiers by covering the topics of model-level neural network design, algorithm-level training optimization and hardware-level instruction acceleration. Aiming at the on-device software and hardware synergy for edge intelligence applications, we hope this survey could guide the researchers to implement high-performance on-device learning systems and further promote the development of edge intelligence techniques.

ACKNOWLEDGMENT

This research was supported by the funding from Hong Kong RGC Research Impact Fund (RIF) with the Project No. R5060-19 and R5034-18, General Research Fund (GRF) with the Project No. 152221/19E and 15220320/20E, Collaborative Research Fund (CRF) with the Project No. C5026-18G, the National Natural Science Foundation of China (61872310), Shenzhen Science and Technology Innovation Commission (R2020A045), Shenzhen Basic Research Funding Scheme JCYJ20170818103849343, and RCN-Diku INTPART BDEM Grant No. 261685.

REFERENCES

- [1] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. of CVPR*, pp. 779–788, 2016.
- [2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," in *Proc. of ECCV*, vol. 9905, pp. 21–37, 2016.
- [3] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, 2011.
- [4] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. of SIGCOMM*, pp. 270–288, 2019.
- [5] N. Sünderhauf, O. Brock, W. J. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, "The limits and potentials of deep learning for robotics," *I. J. Robotics Res.*, vol. 37, no. 4-5, pp. 405–420, 2018.
- [6] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective cluster scheduling for deep learning," in *Proc. of OSDI*, 2018.
- [7] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proc. of ICLR*, 2020.
- [8] X. Wang, F. Yu, Z. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *Proc. of ECCV*, vol. 11217, pp. 420–436, 2018.
- [9] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. of AISTATS*, vol. 89, pp. 1215–1225, 2019.
- [10] N. H. Tran, W. Bao, A. Y. Zomaya, M. N. H. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *Proc. of INFOCOM*, pp. 1387–1395, 2019.
- [11] B. Fang, X. Zeng, and M. Zhang, "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. of MobiCom*, pp. 115–127, 2018.
- [12] M. Tsukada, M. Kondo, and H. Matsutani, "A neural network-based on-device learning anomaly detector for edge devices," *IEEE Trans. Computers*, vol. 69, no. 7, pp. 1027–1044, 2020.
- [13] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.
- [14] T. L. M. for Mobile and E. Devices. <https://www.tensorflow.org/lite>, 2020.
- [15] P. Mobile. <https://pytorch.org/mobile/home/>, 2020.
- [16] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. of ICLR*, 2017.
- [17] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," in *Proc. of NeurIPS*, pp. 4077–4087, 2017.
- [18] S. Dhar, J. Guo, J. Liu, S. Tripathi, U. Kurup, and M. Shah, "On-device machine learning: An algorithms and learning theory perspective," *arXiv preprint*, vol. abs/1911.00623, 2019.
- [19] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *Proc. of ICLR*, 2019.
- [20] N. Lee, T. Ajanthan, and P. H. S. Torr, "Snip: single-shot network pruning based on connection sensitivity," in *Proc. of ICLR*, 2019.
- [21] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. of CVPR*, pp. 4340–4349, 2019.
- [22] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *arXiv preprint*, vol. abs/1506.02626, 2015.
- [23] X. Dong, S. Chen, and S. J. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proc. of NeurIPS*, pp. 4857–4867, 2017.
- [24] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," *arXiv preprint*, vol. abs/1510.00149, 2016.
- [25] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Proc. of NeurIPS*, pp. 1379–1387, 2016.
- [26] L. Breiman and N. Shang, "Born again trees," in *Technical Report*, 1996.
- [27] C. Bucila, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. of KDD*, pp. 535–541, 2006.
- [28] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint*, vol. abs/1503.02531, 2015.
- [29] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," in *Proc. of ICLR*, 2018.
- [30] L. Theis, I. Korshunova, A. Tejani, and F. Huszár, "Faster gaze prediction with dense networks and fisher pruning," *arXiv preprint*, vol. abs/1801.05787, 2018.
- [31] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Hardware-software codesign of accurate, multiplier-free deep neural networks," in *Proc. of DAC*, pp. 28:1–28:6, ACM, 2017.
- [32] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning transferable features with deep adaptation networks," in *Proc. of ICML*, vol. 37, pp. 97–105, 2015.
- [33] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," in *Proc. of CVPR*, pp. 512–519, 2014.
- [34] H. Azizpour, A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson, "Factors of transferability for a generic convnet representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 9, pp. 1790–1802, 2016.
- [35] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proc. of CVPR*, pp. 3994–4003, 2016.
- [36] Z. Yang, J. J. Zhao, B. Dhingra, K. He, W. W. Cohen, R. Salakhutdinov, and Y. LeCun, "Glomo: Unsupervisedly learned relational graphs as transferable representations," *arXiv preprint*, vol. abs/1806.05662, 2018.

- 2018.
- [37] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, B. Catanzaro, and W. J. Dally, "DSD: dense-sparse-dense training for deep neural networks," in *Proc. of ICLR*, 2017.
- [38] H. Zhang, J. Wang, Z. Sun, J. M. Zurada, and N. R. Pal, "Feature selection for neural networks using group lasso regularization," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 4, pp. 659–673, 2020.
- [39] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the regularity of sparse structure in convolutional neural networks," *arXiv preprint*, vol. abs/1705.08922, 2017.
- [40] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, pp. 32:1–32:18, 2017.
- [41] X. Wu, Y. Wu, and Y. Zhao, "Binarized neural networks on the imagenet classification task," *arXiv: Computer Vision and Pattern Recognition*, 2016.
- [42] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," in *Proc. of ICLR*, 2016.
- [43] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. of ECCV*, vol. 9908, pp. 525–542, 2016.
- [44] P. Gysel, J. J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 29, no. 11, pp. 5784–5789, 2018.
- [45] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," in *Proc. of ICLR*, 2017.
- [46] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint*, vol. abs/1606.06160, 2016.
- [47] A. K. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "WRPN: wide reduced-precision networks," in *Proc. of ICLR*, 2018.
- [48] J. Choi, Z. Wang, S. Venkataramani, P. I. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: parameterized clipping activation for quantized neural networks," *arXiv preprint*, vol. abs/1805.06085, 2018.
- [49] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified INT8 training for convolutional neural network," *arXiv preprint*, vol. abs/1912.12607, 2019.
- [50] J. Jiang, F. Fu, T. Yang, and B. Cui, "Sketchml: Accelerating distributed machine learning with data sketches," in *Proc. of SIGMOD*, pp. 1269–1284, 2018.
- [51] N. Ivkic, D. Rothchild, E. Ullah, V. Braverman, I. Stoica, and R. Arora, "Communication-efficient distributed SGD with sketching," in *Proc. of NeurIPS*, pp. 13144–13154, 2019.
- [52] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey, and A. Raghu-nathan, "Scaleddeep: A scalable compute architecture for learning and evaluating deep networks," in *Proc. of ISCA*, 2017.
- [53] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for DNN training," in *Proc. of SOSP*, pp. 1–15, 2019.
- [54] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *Proc. of ATC*, 2017.
- [55] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed DNN training acceleration," in *Proc. of SOSP*, pp. 16–29, 2019.
- [56] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. of NeurIPS*, 2019.
- [57] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *Proc. of MICRO*, pp. 609–622, 2014.
- [58] A. N. Engine. <https://developer.apple.com/machine-learning/>, 2020.
- [59] H. K. NPU. <https://consumer.huawei.com/en/campaign/kirin980/>, 2020.
- [60] S. O. Settle, M. Bollavaram, P. D'Alberto, E. Delaye, O. Fernandez, N. C. Fraser, A. Ng, A. Sirasao, and M. Wu, "Quantizing convolutional neural networks for low-power high-throughput inference engines," *arXiv preprint*, vol. abs/1805.07941, 2018.
- [61] L. Lai, N. Suda, and V. Chandra, "Deep convolutional neural network inference with floating-point weights and fixed-point activations," *arXiv preprint*, vol. abs/1703.03073, 2017.
- [62] C. Wu, M. Wang, X. Li, J. Lu, K. Wang, and L. He, "Phoenix: A low-precision floating-point quantization oriented architecture for convolutional neural networks," *arXiv preprint*, vol. abs/2003.02628, 2020.
- [63] B. McDanel, S. Q. Zhang, H. T. Kung, and X. Dong, "Full-stack optimization for accelerating cnns using powers-of-two weights with FPGA validation," in *Proce. ICS*, pp. 449–460, 2019.
- [64] T. T. Vu, D. T. Ngo, N. H. Tran, H. Q. Ngo, M. N. Dao, and R. H. Middleton, "Cell-free massive MIMO for wireless federated learning," *arXiv preprint*, vol. abs/1909.12567, 2019.
- [65] T. Huang, B. Ye, Z. Qu, B. Tang, L. Xie, and S. Lu, "Physical-layer arithmetic for federated learning in uplink MU-MIMO enabled wireless networks," in *Proc. of Infocom*, 2020.
- [66] H. E. Robbins, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 2007.
- [67] A. J. R. Simpson, "Instant learning: Parallel deep neural networks and convolutional bootstrapping," *arXiv preprint*, vol. abs/1505.05972, 2015.
- [68] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proc. of SIGKDD*, New York, 2014.
- [69] A. F. I. A. Technology. <https://support.apple.com/en-us/HT208108>, 2020.
- [70] M. F. Recognition. <https://www.microsoft.com/en-us/garage/profiles/flower-recognition/>, 2020.
- [71] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proc. of MobiSys*, pp. 68–81, 2017.
- [72] H. Hu, D. Wang, and C. Wu, "Distributed machine learning through heterogeneous edge systems," in *Proc. of AAAI*, pp. 7179–7186, 2020.
- [73] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation of-flooding for multicore-based mobile devices," in *Proc. of INFOCOM*, pp. 46–54, 2018.
- [74] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentral-ized data," in *Proc. of AISTATS*, vol. 54, pp. 1273–1282, 2017.
- [75] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *Proc. of SIGKDD*, pp. 701–710, ACM, 2014.
- [76] A. C. M. Framework. <https://developer.apple.com/documentation/coreml>, 2020.
- [77] F. Manessi, A. Rozza, S. Bianco, P. Napolitano, and R. Schettini, "Automated pruning for deep neural network compression," in *Proc. of ICPR*, pp. 657–664, 2018.
- [78] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," in *Proc. of ICLR*, 2018.
- [79] D. Rothchild, A. Panda, E. Ullah, N. Ivkic, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, "Fetchsgd: Communication-efficient fed-erated learning with sketching," *arXiv preprint*, vol. abs/2007.07682, 2020.
- [80] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of CVPR*, pp. 770–778, 2016.
- [81] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010.
- [82] W. Park, D. Kim, Y. Lu, and M. Cho, "Relational knowledge distilla-tion," in *Proc. of CVPR*, pp. 3967–3976, 2019.
- [83] F. Tung and G. Mori, "Similarity-preserving knowledge distillation," in *Proc. of ICCV*, pp. 1365–1374, 2019.
- [84] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *Proc. of ICLR*, 2015.
- [85] Z. Huang and N. Wang, "Like what you like: Knowledge distill via neuron selectivity transfer," *arXiv preprint*, vol. abs/1707.01219, 2017.
- [86] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via atten-tion transfer," in *Proc. of ICLR*, 2017.
- [87] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results," in *Proc. of NeurIPS*, pp. 1195–1204, 2017.
- [88] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li, "Imagenet: A large-scale hierarchical image database," in *Proc. of CVPR*, pp. 248–255, 2009.
- [89] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [90] R. Li, Y. Wang, F. Liang, H. Qin, J. Yan, and R. Fan, "Fully quantized network for object detection," in *Proc. of CVPR*, pp. 2810–2819, 2019.
- [91] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. of CVPR*, pp. 2261–2269, 2017.

- 2017.
- [92] Q. Chen, J. Huang, R. S. Feris, L. M. Brown, J. Dong, and S. Yan, "Deep domain adaptation for describing people based on fine-grained clothing attributes," in *Proc. of CVPR*, pp. 5315–5324, 2015.
 - [93] W. Ge and Y. Yu, "Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning," in *2Proc. CVPR*, pp. 10–19, 2017.
 - [94] A. Kumar, P. Sattigeri, K. Wadhawan, L. Karlinsky, R. S. Feris, B. Freeman, and G. W. Wornell, "Co-regularized alignment for unsupervised domain adaptation," in *Proc. of NeurIPS*, pp. 9367–9378, 2018.
 - [95] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang, "Convolutional neural networks for medical image analysis: Full training or fine tuning?," *IEEE Trans. Medical Imaging*, vol. 35, no. 5, pp. 1299–1312, 2016.
 - [96] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?," in *Proc. of CVPR*, pp. 2661–2671, 2019.
 - [97] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Proc. of NeurIPS*, pp. 3320–3328, 2014.
 - [98] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. of CVPR*, pp. 6848–6856, 2018.
 - [99] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint*, vol. abs/1704.04861, 2017.
 - [100] A. Achille, T. Eccles, L. Matthey, C. Burgess, N. Watters, A. Lerchner, and I. Higgins, "Life-long disentangled representation learning with cross-domain latent homologies," in *Proc. of NeurIPS*, pp. 9895–9905, 2018.
 - [101] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware distributed parameter servers," in *Proc. of SIGMOD*, 2017.
 - [102] L. Wan, M. D. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proc. of ICML*, vol. 28, pp. 1058–1066, 2013.
 - [103] C. Louizos, M. Reisser, T. Blankevoort, E. Gavves, and M. Welling, "Relaxed quantization for discretized neural networks," in *Proc. of ICLR*, 2019.
 - [104] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. of ICML*, vol. 37, pp. 1737–1746, 2015.
 - [105] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint*, vol. abs/1806.08342, 2018.
 - [106] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. of CVPR*, pp. 2704–2713, 2018.
 - [107] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.
 - [108] M. Greenwald and S. Khanna, "Space-efficient online computation of quantile summaries," in *Proc. of SIGMOD*, pp. 58–66, 2001.
 - [109] E. Andriyash, A. Vahdat, and W. G. Macready, "Improved gradient-based optimization over discrete distributions," *arXiv preprint*, vol. abs/1810.00116, 2018.
 - [110] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. of SIGCOMM*, 2008.
 - [111] A. Animoji. <https://support.apple.com/en-gb/HT208190>, 2020.
 - [112] C. Ding, S. Wang, N. Liu, K. Xu, Y. Wang, and Y. Liang, "REQ-YOLO: A resource-aware, efficient quantization framework for object detection on fpgas," in *Proc. of FPGA* (K. Bazargan and S. Neuendorffer, eds.), pp. 33–42, 2019.
 - [113] J. Yang, K. Zhang, H. Fan, Z. Huang, Y. Xiang, J. Yang, L. He, L. Zhang, Y. Yang, R. Li, et al., "Development and validation of deep learning algorithms for scoliosis screening using back images," *Communications biology*, vol. 2, no. 1, pp. 1–8, 2019.
 - [114] Y. Liu, Z. Li, Z. Liu, and K. Wu, "Real-time arm skeleton tracking and gesture inference tolerant to missing wearable sensors," in *Proc. of MobiSys*, pp. 287–299, 2019.
 - [115] N. A. Architecture. <https://www.nvidia.com/en-us/data-center/nvidia-ampere-gpu-architecture/>, 2020.
 - [116] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. of ASPLOS*, pp. 269–284, 2014.

Qihua Zhou received the B.S. degree in information security from the Nanjing University of Posts and Telecommunications, China, in 2015. His research interests are about distributed systems, parallel computing, on-device learning and domain-specific hardware acceleration.

Zhihao Qu received his B.S. and Ph.D. degree in computer science from Nanjing University, Nanjing, China, in 2009, and 2018, respectively. He is currently an assistant researcher in the School of Computer and Information at Hohai University and in the Department of Computing at The Hong Kong Polytechnic University. His research interests are mainly in the areas of wireless networks, edge computing, and distributed machine learning.

Song Guo (M'02-SM'11-F'19) is a Full Professor in the Department of Computing at The Hong Kong Polytechnic University. He also holds a Changjiang Chair Professorship awarded by the Ministry of Education of China. His research interests are mainly in the areas of big data, edge AI, mobile computing, and distributed systems. With many impactful papers published in top venues in these areas, he has been recognized as a Highly Cited Researcher (Web of Science) and received over 12 Best Paper Awards from IEEE/ACM conferences, journals and technical committees. Prof. Guo is the Editor-in-Chief of IEEE Open Journal of the Computer Society. He has served on IEEE Communications Society Board of Governors, IEEE Computer Society Fellow Evaluation Committee, and editorial board of a number of prestigious international journals like IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Cloud Computing, IEEE Internet of Things Journal, etc. He has also served as chair of organizing and technical committees of many international conferences. Prof. Guo is an IEEE Fellow and an ACM Distinguished Member.

Boyuan Luo received the B.S. degree in Life and Environmental sciences Department from the Guilin University of Electronic Technology, China, in 2017. He is currently working toward Master degree in the Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong. His research interests are about deep learning, model compression and on-device learning.

Jingcai Guo (M'18) received his BEng degree in computer science from Sichuan University, China in 2013 and the MEng degree in computer science from Waseda University, Japan in 2015. He is currently a Ph.D. candidate in the Department of Computing, at Hong Kong Polytechnic University, Hong Kong SAR. He was a visiting researcher at the University of Sydney. His research interests include machine learning, multimedia, computer vision and semi/un-supervised learning. He is a member of IEEE, IEEE Signal Processing Society, and IEEE Computer Society. He has served as technical program committee (TPC) member and referee for several IEEE conferences and journals. He is the recipient of the Hong Kong Ph.D. Fellowship Scheme 2017/18.

Zhenda Xu received the B.S. degree in Optoelectronic Information Engineering Department from the University of Electronic Science and Technology, China, in 2017. His research interests are about medical image processing, on-device learning and domain-specific hardware acceleration.

Rajendra Akerkar is professor and head of Big Data Research Group at Western Norway Research Institute (Vestlandsforskning), where his primary domain of activities is big data and semantic technologies with aim to combine strong theoretical results with high impact practical results. His recent research focuses on application of big data methods to real-world challenges in mobility, transport, energy and emergency management. He is serving as an Associate Editor of International Journal of Metadata, Semantics and Ontologies (IJMSO), Associate Editor of IEEE Open Journal of the Computer Society (OJ-CS) and Knowledge Management Track Editor of Web Intelligence, an international journal. He has authored 16 books, 139 research papers and edited 19 volumes of international conferences and workshops. He is also actively involved in several international ICT initiatives, and research & innovation projects, including H2020 projects, for more than 20 years.