Enabling Resource-efficient AIoT System with Cross-level Optimization: A survey

Sicong Liu, *Member, IEEE,* Bin Guo*, *Senior Member, IEEE,* Cheng Fang, Ziqi Wang, Shiyan Luo, Zimu Zhou, *Member, IEEE,* Zhiwen Yu, *Senior Member, IEEE*

***Abstract*—The emerging field of artificial intelligence of things (AIoT, *AI+IoT*) is driven by the widespread use of intelligent infrastructures and the impressive success of deep learning (DL). With the deployment of DL on various intelligent infrastructures featuring rich sensors and weak DL computing capabilities, a diverse range of AIoT applications has become possible. However, DL models are notoriously resource-intensive. Existing research strives to realize near-/realtime inference of AIoT live data and low-cost training using AIoT datasets on resource-scare infrastructures. Accordingly, the accuracy and responsiveness of DL models are bounded by resource availability. To this end, the algorithm-system co-design that jointly optimizes the *resource-friendly DL models* and *model-adaptive system scheduling* improves the runtime resource availability and thus pushes the performance boundary set by the standalone level. Unlike previous surveys on resource-friendly DL models or hand-crafted DL compilers/frameworks with partially fine-tuned components, this survey aims to provide a broader optimization space for more free resource-performance tradeoffs. The cross-level optimization landscape involves various granularity, including the DL model, computation graph, operator, memory schedule, and hardware instructor in both on-device and distributed paradigms. Furthermore, due to the dynamic nature of AIoT context, which includes heterogeneous hardware, agnostic sensing data, varying user-specified performance demands, and resource constraints, this survey explores the context-aware inter-/intra-device controllers for automatic cross-level adaptation. Additionally, we identify some potential directions for resource-efficient AIoT systems. By consolidating problems and techniques scattered over diverse levels, we aim to help readers understand their connections and stimulate further discussions.**

***Index Terms*—Resource-efficient AIoT system, cross-level optimization, DL inference and training tasks**

I. Introduction

The Artificial Internet of Things (AIoT), also known as *AI+IoT*, was coined in 2017 and quickly gained widespread attention [1]. On the one hand, the rapid development of deep learning (DL) has led to the emergence of numerous intelligent services. On the other hand, the richer sensors and enhanced DL computing capabilities of intelligent infrastructures have given rise to a new category of devices known as *AIoT devices*. AIoT devices are distinct from traditional IoT sensor nodes due to their broader range of sensing capabilities, ability to perform complex computations directly on the device, and greater connectivity capabilities.

Sicong Liu, Bin Guo, Cheng Fang, Ziqi Wang, Shiyan Luo, and Zhiwen Yu were with the Department of Computer Science, Northwestern Polytechnical University, Xi'an, China. Zhiwen Yu was also with the Harbin Engineering University, Harbin, China. Zimu Zhou was with the School of Data Science, City University of Hong Kong. Corresponding authors: Bin Guo (guobin.keio@gmail.com) and Zhiwen Yu (zhiwenyu@nwpu.edu.cn).

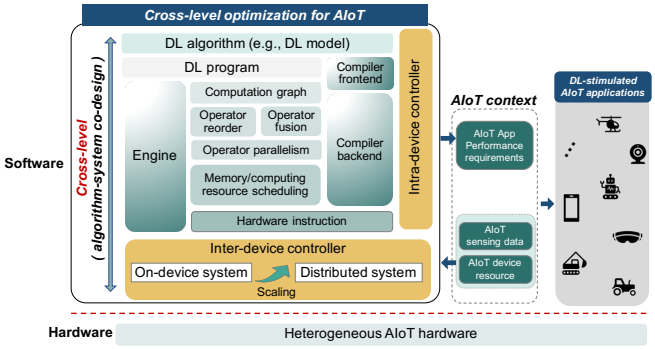


Fig. 1: Illustration of cross-level optimization for the resource-efficient AIoT system, spanning the resource-friendly algorithm, model-adaptive system scheduling, to context-aware intra-/inter-device controllers.

Moreover, there is a growing trend to integrate DL-powered intelligence into tiny embedded AIoT devices for two primary reasons. *First*, the proliferation of AIoT devices has resulted in a massive increase in distributed sensing data captured in various modalities [2]. By executing DL inference and training tasks on resource-scarce AIoT devices, rather than transmitting data to remote centers, *e.g.*, cloud, like traditional IoT, we can save bandwidth and latency while guaranteeing recognition accuracy. *Second*, AIoT applications such as medical assistance and security monitoring collect sensitive user information, posing well-known privacy risks [3]. It is preferable to process data locally or at trusted nearby devices. However, modern DL models are notoriously resource-intensive, making it challenging to achieve real-time inference and low-cost training on resource-scarce AIoT devices.

Given these challenges, previous research has explored *resource-friendly DL models* and *resource scheduling* techniques, either individually or in combination. *First*, *resource-friendly DL model compression* have been widely investigated to reduce the resource demands of DL models at the algorithm level. They include standalone model compression techniques [4], [5], and automated neural architecture search (NAS) frameworks [6]–[8]. However, despite extensive research on model compression, compressed DL models typically compromise accuracy to reduce resource demands. *Second*, some *hand-crafted DL compilers/frameworks* fine-tune diverse system levels to reuse the input data and reduce runtime overhead. For example, TVM [9] optimizes DL operators at the computation graph level, Tensorflow

Runtime (TFRT) [10] implements efficient execution of the computing kernel, and TensorflowXLA [11] designs a linear algebra compiler engine for the TensorFlow framework. *Third*, algorithm-system co-design brings better performance. For example, MCUNet [12] co-designs the TinyNAS, for DL model design, and TinyEngine, for code compilation and memory scheduling. However, algorithm-system co-design is difficult for non-experts. And existing techniques only manually optimize partial system levels to provide a feasible solution.

To this end, providing a relatively complete picture of the cross-level optimization space is necessary. This can assist researchers in finding suitable technique combinations for their specific requirements more freely and help automated frameworks build a finer-grained search space to push the boundary of performance-resource tradeoffs. We summarize the cross-level optimization space as follows:

- *Resource-friendly algorithm level* is concerned with specifying DL models to balance performance and resource constraints. However, the accuracy of lightweight models is degraded and bounded by resource budgets.
- *Model-adaptive system scheduling level* comprises various fine-grained granularities, *e.g.*, computation graph, operator, memory, compiler, engine, and instructor. This level aims at utilizing hardware resources to their fullest capacity without compromising model accuracy.

Joint optimizing across these levels with bi-directional feedback can improve runtime resource availability and performance-resource tradeoff for AIoT. However, no existing surveys have extensively covered all of these levels. The most related previous surveys include [13]–[17]. *First*, many mobile and embedded DL surveys focus on resource-friendly model compression [13], [18] from the algorithm level. *Second*, surveys on edge computing [15], [16] spanning networking and computation offloading [14], [16], pay little attention to the distributed cross-level optimization, *e.g.*, data movement in partitioned computation graphs and memory allocation across devices at runtime. *Third*, [17] maps the DL computation to hardware. However, it targets DL-oriented compiler optimization, which does not co-design the DL models with compilers. This paper comprehensively analyzes the resource-efficient AIoT systems, covering all of these levels. As depicted in Figure 1, it not only encompasses optimization techniques used in stand-alone fields like on-device DL [19], distributed DL [20], [21], and tiny systems [22] but also expands the optimization possibility in the AIoT context.

In particular, the dynamic nature of AIoT context poses a significant challenge: how to adaptively optimize cross-level DL systems to meet varying application performance requirements while satisfying resource constraints. The *dynamic context* includes a range of factors, including heterogeneous AIoT resources, agnostic live data, varying user-specified performance demands, and device-imposed resource constraints. To address these challenges, the AIoT system should also contain context-aware controllers across these levels:

- *Intra-device cross-level controller* establishes a control flow across different system levels to automate the dynamic context awareness, optimization technique combination, and adaptive co-design loop.
- *Inter-device cross-level controller* judges the complementarity of distributed AIoT sensing data and resources, scaling cross-level systems between on-device and distributed schemes.

Therefore, we identify essential enabling technologies for diverse tasks. Each of them encompasses cross-layer optimization, but with different constraints.

- Cross-level optimization for On-device DL inference (§ III-A). It aims to achieve better real-time performance and higher accuracy by minimizing DL model redundancy and maximizing on-device resource capability.
- Cross-level optimization for Distributed DL inference (§ III-B). By aggregating more computing resources and sensing sources, it can further optimize latency and accuracy than the on-device scheme. It operates with a similar cross-level spectrum but utilizes distributed scheduling.
- Cross-level optimization for on-device DL training (§ IV-A). DL training is more complex than inference. It aims to reduce training costs and maintain accuracy.
- Cross-level optimization for distributed DL training (§ IV-B). It further coordinates data fusion and resource aggregation to optimize DL training efficacy and efficiency.
- Resource-efficient AIoT applications (§ V). The aforementioned techniques and systems stimulate a wide range of AIoT applications with flexibility and adaptivity.

In summary, the key contributions of this work can be summarized as follow:

- To the best of our knowledge, this is the first to describe the characteristics and architectures of the resource-efficient AIoT system exactly. It provides a *cross-level spectrum* of the system optimization space for AIoT.
- We propose a novel taxonomy of existing techniques, summarizing how state-of-the-art address issues across different levels of the resource-efficient AIoT system. Additionally, we demonstrate how context-aware controllers can automatically select cross-level techniques for AIoT.
- We discuss open issues in resource-efficient AIoT systems and suggest potential future research directions.

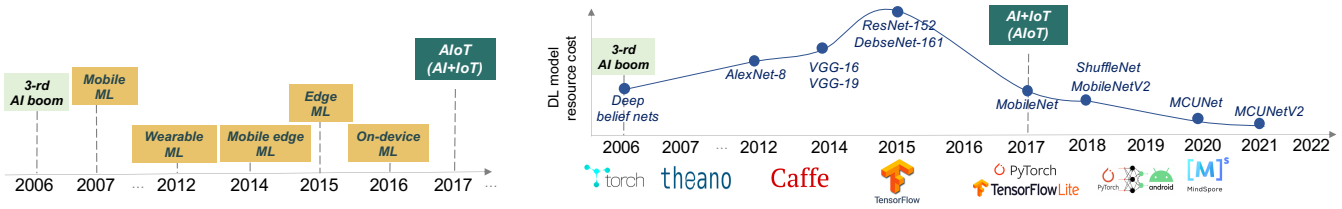
This section introduces the background and leads to the motivation for this survey. In the rest of this paper, we present fundamentals of the resource-efficient AIoT system in § II, introduce the enabling techniques across diverse levels for DL inference and training tasks in § III and § IV, respectively. And then, we list related AIoT systems and applications in § V. Finally, we discuss the open issues in § VI and conclude this paper in § VII.

II. Fundamentals of Resource-efficient AIoT system

This section presents an overview of the resource-efficient AIoT system, departing from existing related areas.

A. AIoT Paradigms

Artificial Intelligence of Things (AIoT) refers to the integration of artificial intelligence (AI) technologies with Internet



(a) Development of related areas

(b) Development of DL models (a showcase of vision fields) and frameworks

Fig. 2: Illustration of various related areas and frameworks.

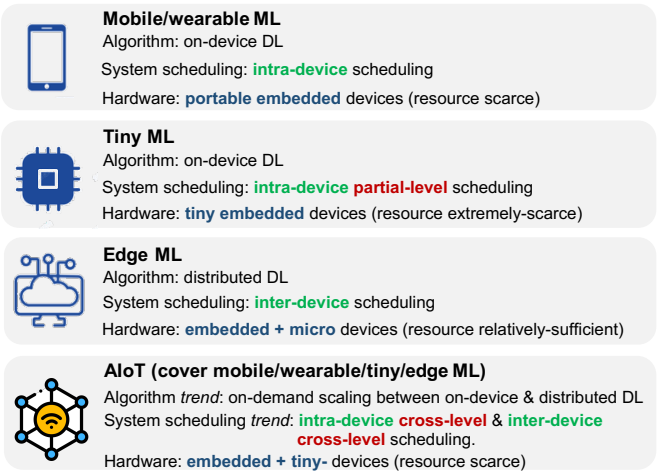


Fig. 3: Difference of related paradigms.

of Things (IoT) infrastructures to improve data analytics [23]. The primary computational task in the resource-efficient AIoT system is *data analysis*. As depicted in Figure 2, deep learning (DL) has emerged as the dominant AI methodology for learning and analyzing data since the third AI boom [24]. And there is a growing trend to incorporate DL-powered intelligence into tiny embedded AIoT devices with the advancement of embedded hardware and on-device DL technologies [25]. In addition, new development frameworks have been launched specifically targeting embedded devices, such as TensorFlow Lite [26], Caffe2 [27], and Pytorch Mobile [28], in order to promote DL-based AIoT applications.

Several related areas utilize overlapping enabling techniques but have varying focuses, as shown in Figure 2a and Figure 3. We differentiate them below.

1) Mobile [29] and *Wearable ML* [30] focus on mobile data analytic patterns and application experience (*e.g.*, real-time response) on *portable embedded* mobiles and wearables.

2) Tiny ML [31] concerns machine learning aware architectures, frameworks, techniques, tools, and approaches which are capable of performing on-device analytics at *tiny embedded* devices with extremely limited resources, *e.g.*, Microprogrammed control unit (MCU). And TinyML project [32] aims to improve the efficiency of systems by requiring less computation, fewer engineers, and fewer data to facilitate the giant market of tiny embedded applications. It covers the management of data and the deployment of models.

3) Edge ML presents to keep data near where it's generated, avoiding costly and privacy-threatening data transfers [33]. Instead of shipping data centrally to perform data analytics, edge computing analyzes data using ML at the edge while maintaining accuracy and latency. The edge includes embedded and micro devices with relatively sufficient resources.

As these related areas evolve, many enabling techniques and frameworks can facilitate AIoT from different aspects. Techniques such as DL models for mobile and edge computing [30] and on-device/distributed DL deployment [29], [33] provide various model compression and offloading algorithms for inference and training. Frameworks such as TFlite [26], CMix-NN [34], TVM [9], TensorFlow XLA [11], and oneDNN [35] offer a range of acceleration options and memory scheduling support for DL. The key distinction lies in the AIoT system's ability to integrate these technologies in a novel, context-aware and cross-level manner. Specifically, we define some important concepts as follows:

AIoT computing task and paradigm. *AIoT data* mainly includes two types, *i.e.*, *live sensing data* and accumulated *dataset*. And their analysis is concentrated in two stages, *i.e.*, DL *inference* and *training*, respectively.

- *Near-/realtime inference of AIoT live data on resource-scare AIoT devices*. The AIoT *live data* are sensed by distributed AIoT devices and should be analyzed fastly.
- *Low-cost training using AIoT dataset on resource-scare AIoT devices*. The AIoT *dataset* are sensed and held by distributed AIoT devices and should be consumed with low data transmission and training costs, *e.g.*, memory.

Alternatively, inference and training tasks can be performed on-device or at distributed edges within the networked system, which enable a plethora of innovative AIoT applications that go beyond the conventional IoT paradigm in terms of accuracy, latency, bandwidth, privacy, and energy efficiency [1].

AIoT devices refer to intelligent end/edge devices equipped with advanced sensors and DL computing capabilities. This category includes mobiles, wearables, robots, drones, and other physical devices. In this context, the "end" device serves as the primary sensing source. Similarly to edge computing, the "edge" refers to devices located between the sensing data source and the cloud center path. In the realm of AIoT, there is a preference for prioritizing physically nearby edge devices to enable cost-effective near-sensing-data computing.

Resource-efficient AIoT system. Unlike software-hardware co-design approaches [36], [37], our focus is on the system software, specifically the ***algorithm-system co-design***. This approach allows for effective utilization of existing hardware

resources in dynamic AIoT contexts. AIoT applications like smart home and smart retail rely on a plethora of AIoT devices equipped with advanced sensors and embedded processors. Replacing these devices with newer hardware can often be costly and impractical. Therefore, adopting a strategy that optimizes the usage of available hardware resources can be an efficient and cost-effective solution in such scenarios. We refer to the DL-based system on AIoT devices as ***resource-efficient AIoT system*** in the following part for short. The resource-efficient AIoT system is responsible for distributing and federating AIoT sensing data analytics across resource-constrained and heterogeneous end/edge devices for DL inference and training. It is worth noting that communication networking for AIoT is outside the scope of this survey.

B. Cross-level Characteristics of AIoT System

We provide an overview of the remarkable cross-level characteristics and architectures of resource-efficient AIoT systems. As illustrated in Figure 4, the resource-efficient AIoT system comprises two key levels: the *resource-friendly algorithm* and *model-adaptive system scheduling*. These levels should be *co-designed* to ensure coherent resource usage. We outline their specific features and functionality below.

1) Resource-friendly algorithm level: dynamically scalable, divisible, and composable DL models. The DL models for AIoT should be *dynamically scalable, divisible, and composable* in both DL inference and training tasks. *First*, the DL model (*e.g.*, structure, parameter size) should be *scalable* with diverse compression degree to satisfy the platform-imposed dynamic resource constraints (*i.e.*, memory, computing, and battery) and application-specified performance demands (*i.e.*, accuracy, latency, energy cost). This scheme will inevitably bring accuracy fluctuations if the DL models are poorly designed. *Second*, DL models' divisible and composable properties are necessary to offload computation to distributed AIoT devices. Particularly, the divisibility of DL models depends on the internal dependency of DL layers, channels, and operators. And the composable property of DL models is able to adjust the system to offload different DL block combinations to diverse AIoT devices for dynamic resource availability.

2) Model-adaptive system scheduling level: maximizing runtime hardware capability. This level aims to utilize hardware resources to their fullest capacity without compromising model accuracy. Even for identical DL model configurations, mapping different model layers/operators onto diverse memory units in varying sequences results in different latency and resource overhead [9], [38], [39]. For example, integrating the memory fragmentation in the tensor layout of SqueezeNet can reduce 42% wasted memory fragmentation [40]. Therefore, designing appropriate strategies at computation graph, operator, and memory allocation levels to cater to the upper-level models can enhance runtime resource availability. Notably, this level can *iteratively allow a more flexible DL model design space for the algorithm level, thereby pushing the limitations on accuracy-resource trade-offs*.

3) Intra-device controller: automating the adaptive on-device cross-level optimization. The efficacy of various optimization techniques at different levels can vary for the same

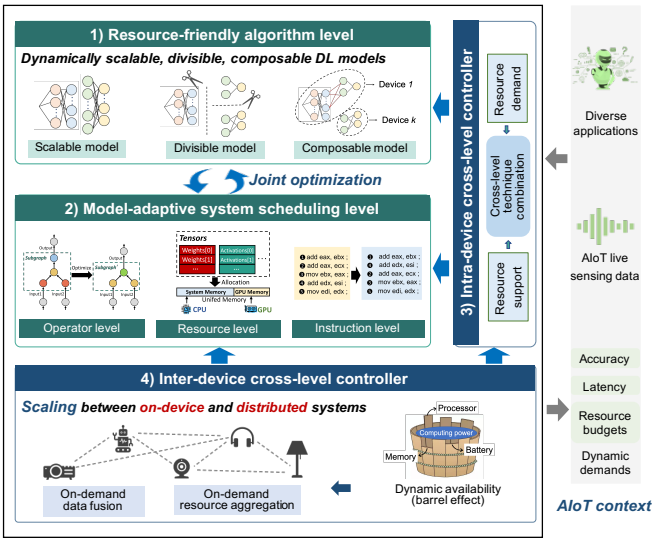


Fig. 4: Resource-efficient AIoT system architecture, involving two joint-optimized levels and two context-aware controllers.

DL model. And even within the same level of optimization techniques, there can be differences in their performance. As a result, it is imperative to develop an extra control flow to automate the adaptive optimization of DL models and system scheduling in a cross-level manner. Also, adaptively adjusting the cross-level techniques based on dynamic contexts, such as input data, resource availability, and user demands, is necessary. The controller monitors the *resource availability* of the target platforms and predicts the *resource requirements* of the AIoT system based on the current model configurations and scheduling strategies. The controller automatically adjusts techniques across different levels if the resource demand exceeds the supply or does not align with the user-defined budgets.

4) Inter-device controller: automating the adaptive distributed cross-level optimization. It scales the cross-level AIoT system from on-device to distributed schemes for achieving better performance-resource efficiency trade-off. As shown in Figure 5, the distributed AIoT devices collaborate on demand for two motivations:

- ***On-demand sensing source association***. With the proliferation of data-rich sensors (*e.g.*, cameras, LIDAR, and hyperspectral imagers), different distributed sensing sources have temporal connections for specific DL training/inference tasks. Distributed multi-modal data benefit the environment/object recognization from different vantage points with various physical properties [41]–[44]. The inter-device controller needs balance the necessity of sensing source association within the networked system to the accuracy of AIoT tasks and the overhead.
- ***On-demand computing resource aggregation***. As mentioned above, AIoT prioritizes the on-device scheme, followed by the distributed collaboration scheme with nearby edges, to realize near-sensing-data computing for saving transmission bandwidth and protecting data privacy [45], [46]. Executing DL models requires large computing/memory resources that are not always avail-

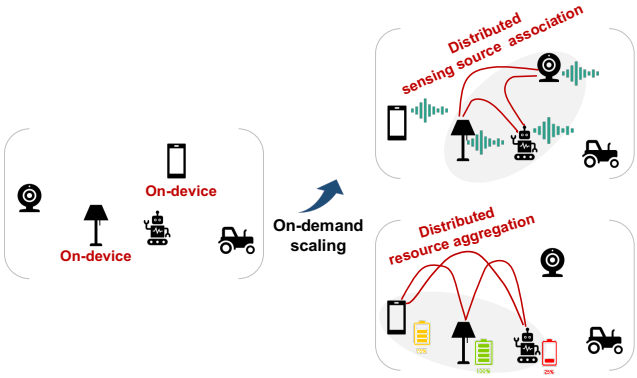


Fig. 5: The inter-device controller scales the cross-level AIoT system between on-device and distributed schemes for on-demand sensing source association or resource aggregation.

able in a single AIoT device, significantly when the scale and complexity of DL models continuously increase. Scheduling the most suitable distributed devices within locally connected and resource-constrained edge clusters is necessary yet challenging [47], [48].

C. Taxonomy of Enabling Techniques

We identify the following essential enabling technologies for resource-efficient AIoT systems, considering the cross-level deployment issues mentioned earlier. Figure 6 summarizes our taxonomy, *i.e.*, cross-level optimization for both resource-efficient *DL inference and training tasks*.

1) Cross-level Optimization for DL Inference Tasks: There is a growing trend today to bring DL-powered intelligence into AIoT devices for various applications, *e.g.*, object recognition [49]–[52], semantic segmentation [53]–[55], object tracking [56]–[58], natural language processing [59], [59], [60], and recommendation [61]–[63].

Challenges. It is non-trivial to achieve near-/real-time DL inference with limited resources in AIoT devices, which is critical for AIoT live data to satisfy the applications' responsiveness. First, on-device DL inference benefits user privacy and robustness. However, mainstream DL models, such as Zero-DCE for low-light video enhancement [64], are still computation-intensive and fail to achieve real-time processing on the local AIoT device. Second, distributed DL inference strives to satisfy stringent demands across multiple dimensions, *e.g.*, latency, accuracy, and transmission/resource cost. Moreover, energy savings in DL inference is also crucial for long-term running applications since most mobile AIoT devices are battery-powered [65], [66]. Also, it is desired to adapt the inference accuracy according to the resource availability and network condition for AIoT at runtime. We detail different levels of issues in the following part.

On-device DL Inference. Prior efforts explored several technologies to enable on-device DL inference, accelerate inference, and save memory occupation or energy. ***(i) resource-friendly algorithm level*** compresses DL models to reduce the resource demand without significantly compromising their accuracy. Standalone techniques include pruning [67], [68], [69], low-rank decomposition [70], lightweight architecture replacement [8], [12], [71], and parameter/activation quantization [72], [73]. Some research [74] also automatically combines diverse compression techniques to achieve better performance-resource tradeoffs. Although significant progress has been made in this field, compressed models typically yield accuracy degradation. ***(ii) model-adaptive system scheduling level*** spans multiple fine-grained levels, *i.e.*, computational graph [75]–[77], memory scheduling [12], [78], hardware instruction [79], [80], compiler front-end/back-end [76], [77], [81], and engine [12]. For example, existing DL frameworks, *e.g.*, Tensorflow [82], Pytorch [83], and TVM [9], provide support in computation graph and operator optimization. IOS [75] extensively studies the inter-operator parallelism to accelerate inference. Miao *et al.* [78] proposed dynamically swapping data between MCU's micro-SRAM and external flash to save SRAM. The suitable underlying scheduling can further improve resource availability than the algorithm-level compression models. However, most existing techniques are manually designed. ***(iii) intra-device cross-level controller*** aims to adaptively select the above-mentioned cross-level optimization techniques according to the user-specified performance demands and the device-imposed resource budgets. For example, AdaDeep [74] is an automated DNN compression framework at the algorithm level that uses deep reinforcement learning to balance performance and resources. MCUNet [12] performs joint optimization across the algorithm and engine levels, adapting the optimization strategies according to memory constraints. However, the combination criteria of cross-level techniques from a broader space remains a black box. ***(iv) inter-device cross-level controller*** will scale up the system from the on-device to distributed scheme once the large computing/memory resources required by executing computation-intensive models locally are unavailable.

Distributed DL Inference. Deploying different parts of DL models on multiple AIoT devices can harness the collective computational power of these devices, thereby decreasing local resource demands and enhancing inference efficiency. ***(i) resource-friendly algorithm level*** involves DL model partition [84], offloading [85], and performance tradeoff [85], [86]. According to the operator dependency of DL models and the resource availability of edge devices, various parts of DL models can be distributed to multiple devices for either sequential or parallel processing, resulting in different performance tradeoffs. For example, [86] jointly carries out the model partition and offloading to improve accuracy, energy consumption, and latency. ***(ii) model-adaptive system scheduling level*** includes computation graph partition [87], [88], distributed operator fusion [89], [90], separately data reuse and memory allocation. For example, Modnn [87] significantly speeds up DL inference by introducing execution parallelism among multiple devices [88]. Zhang *et al.* [91] proposes jointly optimizing heterogeneous chips' computing frequency, power, and memory to achieve the optimal allocation strategy on distributed devices. This area is relatively less explored. ***(iii) inter-device cross-level controller*** has a similar cross-level technique spectrum as the on-device inference scheme. And the fundamental difference lies in the fact that the underlying

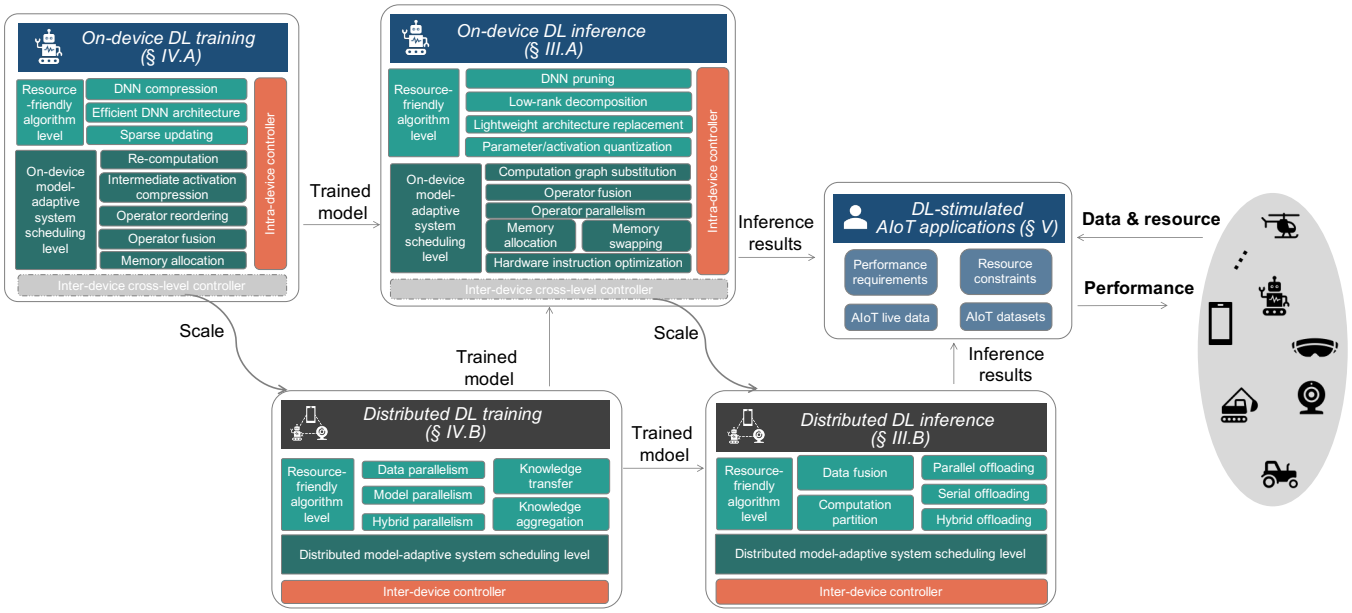


Fig. 6: Landscape of the resource-efficient AIoT system according to the proposed technique taxonomy.

operator and resource scheduling for distributed schemes must be *optimized separately and verified globally*. This is because different parts of the model deployed on multiple devices have separate memory pools and can not reuse data, necessitating separate optimization at diverse devices. Meanwhile, global evaluation is required for overall performance, such as the total latency of distributed execution and transmission. In addition, the controller selects the most suitable devices within the closely connected and resource-constrained edge. It involves the performance profiler [92], adaptive inference serialization or parallelism [93], and autonomatic optimizer [94].

2) Cross-level Optimization for DL Training Tasks: There are many demands for DL training on resource-scarce AIoT devices. For instance, we may need to update pre-trained DL models locally or nearby when the live sensing data drift to the original training data and an Internet connection to the cloud is unavailable. Other requirements for DL training on AIoT devices are also widespread, such as updating models to adapt to new applications. These requirements can be fulfilled using techniques like transfer learning [95] [96] [97] [98], domain adaptation [99] [100] [101] [102], continuous learning [103] [104] [105], and personalized federated learning [106] [107]. Besides, DL model fine-tuning is necessary after adaptive compression, which has been demonstrated in various practical scenarios [108] [109] [110] [111].

Challenges. It is intractable to realize low-cost DL training on AIoT devices. The reasons are three-fold: *i) resource constraint*. The bottleneck of embedded resources in AIoT devices is the memory access bandwidth [112]. While DL training needs *batched memory* chunks grouping multiple data samples for feature learning. Besides, the computation efficiency will be low if the memory for sufficient batch size cannot be secured. Because computations are highly sensitive to memory access schemes, *e.g.*, Cache hit rate [113]; *ii) irregular activation lifecycle*. The DL training phase involves forward propagation and backpropagation to update the model weights iteratively. Intermediate activations pose high memory demands, produced during the forward pass and reused during the backward pass [114]. In the case of DL inference using only forward propagation, resources occupied by activations can be directly released. However, during DL training involving both forward and backpropagation, activations must be retained throughout the process. Model structures, such as control flow and branching, affect the lifecycle of activations during backpropagation, which is less regular. Therefore, it is difficult to determine when data will be accessed for the last time and when it is safe to release resources. *(iii) multiple iterations.* Unlike DL inference's "one-time" property, DL training requires optimization across multiple iterations. Therefore, even small instabilities can be amplified over hundreds of iterations, potentially leading to the crash of DL training [38]. Additionally, current cloud-based DL training optimization techniques are not suitable for resource-scare AIoT devices.

On-device DL Training. Given these challenges, we summarize the enabling techniques in § IV-A that ensure sufficient resource supply and guarantee performance across different system levels. ***(i) resource-friendly algorithm level*** aims to reduce resource demands, especially memory usage, through model or training simplification. Techniques include model quantization [115], model compression [115]–[117], sparse updating [118], [119], *etc.* For example, TinyTL [112] presents the element-wise convolution decomposition to reduce memory, not parameters, for efficient on-device DL training. ***(ii) model-adaptive system scheduling level*** mainly optimizes three objects, *i.e.*, *intermediate activation*, *computation graph*, and *memory schedule*. Precisely, to trim down the *intermediate activation tensor* after the forward and before the backward pass, researchers present the recomputation [120], [121] and activation compression [122] techniques. They discard or compress the intermediate activation tensors to reduce the peak memory during DL training. Optimization techniques at the *computation graph* level include operator reordering [123] and

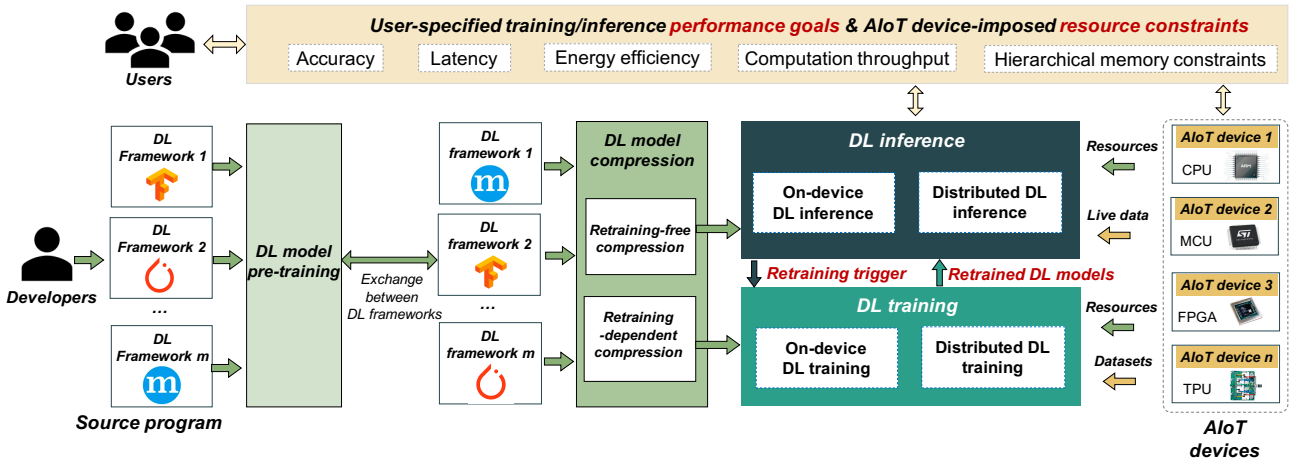


Fig. 7: Workflow of the AIoT system software for resource-efficient DL training and inference tasks on AIoT devices.

operator fusion [124], [125]. Rearranging or fusing operators in the computation graph can reduce memory usage and access delay. As for the *resource scheduling*, prior efforts mainly operate on memory, which is the lowest level optimization closest to hardware, including memory allocation [114] and memory swapping [39], [126]. For example, MoDNN temporarily swaps intermediate activation from graphics processing unit (GPU) memory to host memory to solve the problem of insufficient training memory and thereby realizes the optimal compromise between memory footprint and training speed [126]. ***(iii) intra-device cross-level controller*** jointly control multiple techniques across the above levels to achieve the best tradeoff between multiple conflicting performance goals for dynamic AIoT context. For example, adaptive precision training in [127] dynamically allocates model precision to balance training energy cost, memory usage, and accuracy.

Distributed DL Training. To coordinate data fusion and resource aggregation to optimize DL training efficacy and efficiency on distributed devices. We introduce cross-level enabling techniques for distributed training in § IV-B. In particular, ***(i) resource-friendly algorithm level*** optimizes the distributed training algorithm (*e.g.*, pipeline algorithm [128] and federated learning [129]) and changes the DL model/data partition strategy, which can significantly improve the distributed system's computing efficiency. ***(ii) model-adaptive system scheduling level***, like the on-device training, mainly optimize operator and memory access for training speedups and resource usage reduction, such as memory reallocation [130] and layer swapping [131] *etc.* For example, Zico [132] monitors the memory usage of each DL training task and reclaims the memory that is no longer needed, making it globally sharable in the system. ***(iii) inter-device cross-level controller*** considers all cross-level factors like the on-device DL training scheme. In addition, it considers the communication conditions, such as time-varying network throughput, to control the distributed system loop jointly. And it schedules the cross-level techniques separately and evaluates them globally. Specifically, it selects available devices to minimize overall training delay, ensure accuracy, and improve resource efficiency [133]. To address issues such as asynchronous processing and waiting times resulting from resource-heterogeneous AIoT devices, the inter-device controller should monitor their heterogeneity in advance and select suitable devices for collaboration.

D. Workflow Overview

Figure 7 showcases the resource-efficient AIoT system workflow and relationships of the system blocks for DL training and inference tasks. The system takes user-provided inputs, such as DL source programs, and aims to achieve user-specified performance goals while satisfying the resource constraints imposed by the AIoT devices. Specifically, the *user/developer* specifies the *DL source program* using various DL frameworks (*e.g.*, TensorFlow, Pytorch, Mindspore, *etc.*) and conducts model training using accumulated AIoT dataset. After DL pre-training, the model structure files are distributed to heterogeneous AIoT devices to analyze live data and provide intelligent services based on the pre-trained model. To be compatible with different DL frameworks in various AIoT devices, we can convert them to a unified format and exchange them (*e.g.*, using ONNX [134]). In the *DL inference phase*, we can select and combine the most suitable DNN compression techniques from the *model compression algorithm pools* to reduce the model complexity and resource demands. Most model compression techniques need to return to the training stage for several rounds of retraining to fine-tune model parameters to ensure accuracy [135]. And some recent efforts also realize the retraining-free model compression techniques, *e.g.*, through ensemble training of super-nets [136].

The resource-efficient *DL inference* includes on-device (§ III-A) and distributed (§ III-B) schemes. Particularly, it includes the runtime compilation front-end optimization (target platform-independent), the compilation back-end, and hardware instruction optimization (target platform-dependent). Both the compiler front-end and the back-end belong to the model-adaptive system scheduling level mentioned in § II-B. They convert the DL models designed at the algorithm level into intermediate representations, *i.e.*, computation graph, for further optimization. Compiler front-end optimization focuses on platform-independent optimization, such

as constant folding [137] [138] and common subexpression optimization [139]. And compiler back-end optimization focuses on platform-dependent optimization, such as operator fusion [140] [124], and memory allocation [141] [114].

When the accuracy of the DL model drops below a certain threshold due to changes in application scenarios, data, performance requirements, *etc.*, the *DL training* block is triggered to retrain and update the model. Whether the DL model is re-trained locally on a single AIoT device or on multiple locally-connected yet resource-constrained edge devices depends on how well the supply of device resources (*e.g.*, memory) matches the computing requirements for training and the desired training speed. Once the inter-device controller selects the training scheme, we can proceed to the on-device DL training optimization block (see § IV-A) or the distributed DL training block (see § IV-B). As a separate note, in resource-efficient AIoT system, the compiler optimization is cross-device [82] [9] on heterogeneous and distributed AIoT devices, *e.g.*, from GPU-based edge servers to MCUs, to support intelligent inference/training tasks. The inter-/intra-device controllers adaptively control the cross-level optimization strategies according to context information.

E. AIoT Performance Metrics

The resource-efficient AIoT system in both DL training and inference tasks needs to optimize the user-specified performance goals (*i.e.*, accuracy, latency, energy efficiency, computation throughput) and satisfy device-imposed resource constraints (*i.e.*, hierarchical memory, processor, battery). We listed the related metrics below:

1) Accuracy. The DL model should be accurate enough to guarantee a high-quality AIoT task. The model weights at different scales are well-trained to represent the generic information of recognition objects [142].

2) Memory. The parameters and activations of DL models should be appropriately sized to fit into the memory units of AIoT devices. We can directly calculate the memory needed to run a DL model using the total number of bits associated with weights and activations. And we should optimize how the operations access the data fetched from different memory hierarchies (*e.g.*, dynamic random access memory (DRAM), static random access memory (SRAM)) and how the computation is executed for latency or energy efficiency optimization [8].

- *Memory budgets*. The memory budget is a tough constraint; it decides whether a specific AIoT device can perform the DL inference or training tasks.
- *SRAM utilization*. Improving SRAM utilization and reducing DRAM transmission times can improve computing efficiency [143].
- *Cache/register hit rate* measures the percentage of times that the system is able to retrieve data from the cache in the central processing unit (CPU)/GPU [144] and register in MCU [145], instead of accessing it from the main memory. Higher cache/register hit rates mean that the system can access data more efficiently. We can use it to estimate the system efficiency in the presence of time-varying memory resources via run-time measurement [146].

3) Computational cost. It affects the AIoT system's latency and energy efficiency. We can model the computational cost of DL inference and training tasks as the total number of DL models' multiply-accumulate (MAC) operations.

4) Latency. The complexity of the DL model for inference and training should be controllable to meet diverse user demands on latency. The latency of DL inference/training tasks executed in AIoT devices strongly depends on the given device's architecture and memory hierarchy [147]. And we can refer to the latency predictor, such as nn-Meter [148], [149], for platform-aware latency estimation.

5) Energy efficiency. It is an important metric for battery-powered AIoT devices. Researchers usually formulate it offline and estimate it online since it is prohibitive to directly connect to energy monitors for measurement when the device is in service. The estimation methods include two types:

- *Estimation function*. The energy consumption of DL inference includes computation cost and memory access cost. The former can be formulated as the total energy cost of the total MACs, *i.e.*,, $E_c = \epsilon_1 C$, where ϵ_1 and C denote the energy cost per MAC operation and the total number of MACs, respectively. The latter depends on the storage scheme when executing DL models on the user-given embedded device. And the energy cost of fetching intermediate activations, *i.e.*, memory access, dominates in the DL inference phase.
- *Arithmetic intensity*. The hardware-aware metric, *e.g.*, arithmetic intensity, can proxy the degree of reuse of parameters and activations and the energy cost required for processing inputs [150], [151].

As a separate note, the widely used parameters number, MAC amount, or speedup ratio is not a good approximation for hardware efficiency (*e.g.*, energy cost, latency), which heavily depends on the underlying memory movement and bandwidth bound [8]. For example, Jha *et al.* [151] reported that although SqueezeNet [152] has $51.8\times$ fewer parameters than AlexNet, while it consumes 33% more energy due to its larger amount of activations and data movement. We identify that merely cutting down the parameter size may lead to an increase in activation size, which, in turn, increases the memory footprint, latency, and energy consumption [150].

The dynamic deployment context in continuously running AIoT applications often results in high levels of unpredictability and variability in terms of performance demands on the above metrics. This context includes factors such as agnostic AIoT sensing data, time-varying resource availability, dynamic join and exit of AIoT devices, and fluctuating inference/training request frequency triggered by real-world requirements. Thus, the resource-efficient AIoT system should continuously evolve to balance these metrics in a context-adaptive manner [8], [153].

III. Cross-level Optimization for DL inference

This section introduces existing efforts associated with the proposed challenges in cross-level AIoT systems and highlights how they have addressed some of them.

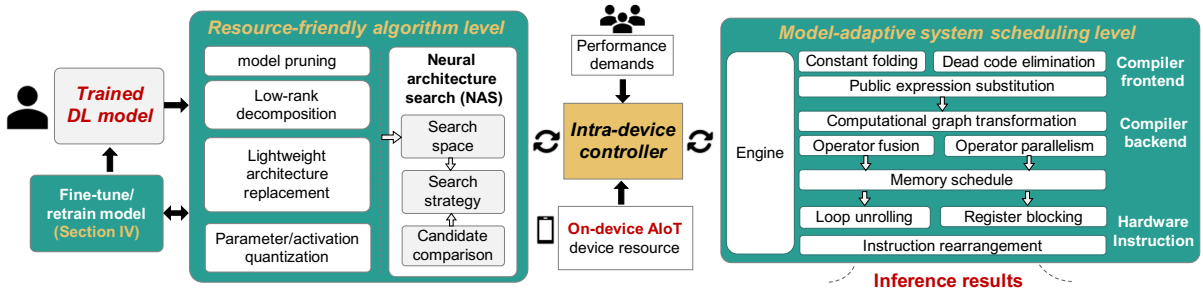


Fig. 8: System loop of the algorithm, system scheduling, and intra-device controller for on-device DL inference.

A. On-device DL Inference

An increasing trend in the field of AIoT is integrating DL-powered intelligence into local devices, allowing for analytics to be performed where the data is sensed. This approach offers advantages such as low transmission costs, network condition independence, and privacy preservation, as highlighted in [154], [155]. However, deploying DL models on resource-scarce local devices remains a challenging problem, as discussed in § II-C. To address this issue, cross-level optimization spans multiple levels (*e.g.*, model, computation graph, memory scheduling, hardware instruction, *etc.*) is essential. Additionally, context-aware controllers can further automate the on-device DL inference process. Figure 8 illustrates the on-device DL inference optimization pipeline.

1) Resource-friendly algorithm level: Algorithm-level optimization for on-device DL inference is extensively researched to minimize computation and memory requirements while preserving accuracy [154], [155] [156]–[161]. We briefly discuss some representative ones below.

a. Pruning. It reduces the model computation cost by removing redundant parameters [162], channels [69], or connections [163]. According to the grain size of pruning, it can be divided into synaptic pruning, neuron pruning, *etc.* Synaptic pruning cuts down unimportant connections between neurons, while neuron pruning removes neuron nodes directly. For example, Hu *et al.* [67] iteratively prunes neurons and uses the average percentage of zeros to find the unimportant activation. [68], [69] determine which channels must be pruned by minimizing the feature reconstruction error using greedy and Least Absolute Shrinkage and Selection Operator(*i.e.*, LASSO) regression optimizers, respectively.

b. Low-rank decomposition. Since model weight vectors are mostly distributed in low-rank subspaces, we can only use a few basis vectors to represent the convolution kernel for memory savings by combining dimensions or applying low-rank constraints [164], [165]. As shown in Figure 9,W_i is a large low-rank matrix which is decomposed into several small matrices such as $W_i^{(1)}, W_i^{(2)}, W_i^{(3)}, ..., W_i^{(k)}$.

Pavel Kaloshin [70] decomposed the tensor as a sum of low-rank and sparse components, approximating the convolution weights. The convolutional (conv) layers take the most execution time, and fully connected (fc) layers dominate storage costs. Lin *et al.* [166] jointly accelerates conv layers and compresses fc layers by utilizing low-rank decomposition to eliminate redundancy between conv kernels and fc matrices.

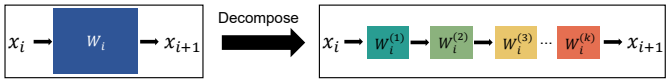


Fig. 9: Illustration of low-rank decomposition technique.

c. Lightweight architecture. Replacing large model blocks with lightweight architectures can adapt to resource-constrained AIoT devices. There are generally two categories, *i.e.*, block replacement [167] or neural architecture search (NAS) [168]. For example, Iandola *et al.* [152] replaced conv layers by Fire block, composing of a 1×1 conv layer and a conv layer with mixed 1×1 and 3×3 filters. Lin *et al.* [169] replaced conv by a micro multi-layer perceptron embedded with multiple small kernel conv (Mlpconv). To realize efficient DL inference on MicroController Unit (MCUs), with 2-3 orders of magnitude smaller memory than mobile phones, Lin *et al.* proposed MCUNet [12], Edgar *et al.* built μNAS [71] to specialize model architectures for MCUs automatically.

d. Parameter/activation quantization. Quantization [170], [171] refers to representing 32-bit floating-point model parameters with relatively low widths, including weight [172], activation value [173]. The model parameters can be unified with layer-wise low-bit-widths (*e.g.*, 16-bit, 8-bit, 2-bit, 1-bit, *etc.*) to reduce memory usage significantly, speed up computing, and reduce power cost. To adapt to the varying memory and computational limitations, Manuele *et al.* [72] modeled the DL inference graph by pure integer operation using mixed low-bit-width quantization. Under specific RAM and FLASH memory constraints in MCUs, Manuele *et al.* [73] use reinforcement learning to pick the best uniform quantization bitwidths for model weights and activations.

Discussion. Algorithm-level optimization techniques have shown great promise in reducing computation and memory requirements with slightly decreased accuracy. However, the degree of accuracy decrease is bounded by the runtime resource availability. While proposing new algorithms may lead to a slight improvement in performance-resource tradeoff,*e.g.*, latency, over existing techniques, their suitability to AIoT also depends on the system scheduling on the given hardware.

2) Model-adaptive system scheduling level: Joint optimization of system scheduling and upper-level algorithms has the potential to overcome performance bottlenecks. This level

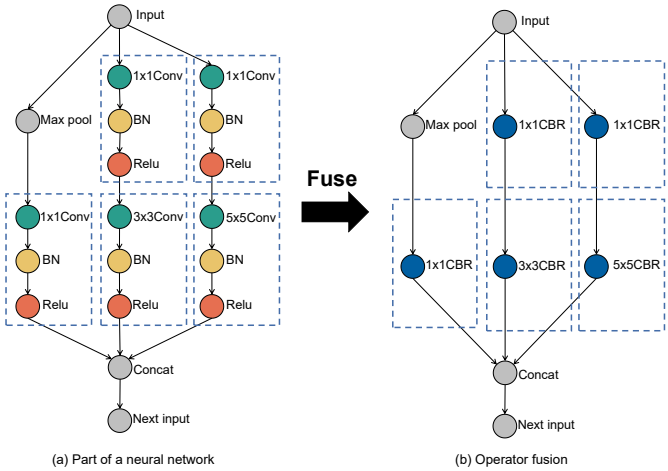


Fig. 10: Illustration of the operator fusion technique. (a) partial computation graph of GoogleNet, (b) longitudinally fuse *convolution, batchnorm* and *relu* into *CBR* operator.

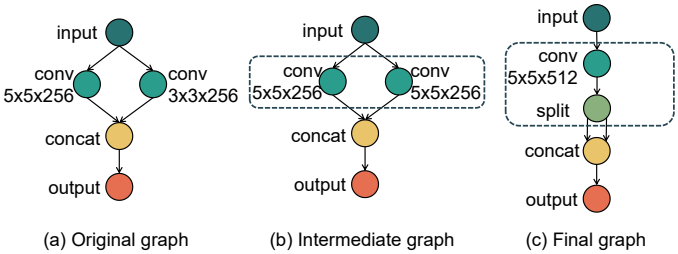


Fig. 11: Illustration of the computation graph substitution technique. (a) original computation graph includes *convolution* operators, (b) expanding smaller conv kernels, and (c) merging two conv operators into one.

aims to utilize hardware resources to their fullest capacity without compromising model accuracy. The system scheduling for DL inference mainly includes the compiler front-end (*i.e.*, device-independent) and compiler back-end (*i.e.*, device-dependent). Compiler front-end optimization aims to eliminate redundancy and simplify the computation of the intermediate representation (*i.e.*, computation graph) during compilation. It is device-independent and contains constant folding [174], dead code elimination [174], *etc.* And compiler back-end aims to use hardware resources to execute DL inference tasks fully. It mainly focuses on computation graph-level optimization (*e.g.*, operator fusion [81], computation graph substitution [124], [175]), operator parallelism [75], [176], memory-level optimization (*e.g.*, memory allocation [177], [178], memory swapping [78], [179]), and hardware instruction optimization (*e.g.*, loop unrolling [180], register blocking [80]). We introduce some representative enabling techniques below.

a. Operator fusion. Despite the massive size of model parameters, the intermediate feature maps extracted from the input data can quickly become too large and consume significant amounts of memory. This is especially problematic because these intermediate feature maps are often used as inputs for multiple operators in the computation graph, leading to increased memory access and processing delays. One solution to this problem is to fuse adjacent operators in the computation graph into a new operator according to certain rules. Figure 10 illustrates an example in GoogleNet. The original operators include convolution, batchnorm, relu, max pool, and concat, as shown in Figure 10a. Then *convolution, batchnorm* and *relu* operators are fused into the *CBR* operator. They have the same computation results. However, the number of model layers in Figure 10b is decreased, the data channel is shortened, and the memory access frequency of the intermediate feature maps is reduced, so the inference speed is improved. Han *et al.* [181] expand the CBR operator fusion to TensorRT.

Existing DL frameworks such as TensorFlow Lite [26], TVM [9], MNN [182], and Pytorch-Mobile [28] have provided application programming interfaces (APIs) for operator fusion. However, most of them only provide fixed operator fusion patterns, and the operator fusion types are still insufficient to support diverse DL operators and connections. Niu *et al.* [77] proposed a universal fusion framework called DNNFusion. It divides operators into different types according to their input and output forms, develops operator fusion plans after comparing the performance of diverse fused operators, and conducts extra optimizations such as reducing redundancy during the fusion code generation. This approach based on general operator type greatly expands the operator fusion opportunity. Experimental results show that the operator fusion space is expanded by 8.8×, and the inference speed exceeds the advanced frameworks (*e.g.*, MNN [182], TVM [9], Pytorch [176], TensorFlow Lite [26]) by up to 9.3×. Another problem is that the operator fusion space and memory efficiency lack direct mapping, making it difficult to access optimal memory. Cai *et al.* [76] proposed an operator fusion framework, Optimus, based on directed acyclic graphs. It can be applied to several DL models running on accelerators. Experiments show that Optimus reduces inference memory access overhead by 17-75% and improves efficiency by 1.86-3.66×. Furthermore, automated operator fusion, instead of manually crafted fusion, can greatly enhance the performance of complicated or previously unseen operator chains.

b. Computation graph substitution. Computation graph substitution techniques replace the subgraph with another functionally equivalent subgraph to reduce the amount of computation and delay. For better understanding, Figure 11 shows an example. In the original graph (see Figure 11a), there are two *conv* operators that have 256 kernels with 3×3 size and 256 kernels with 5×5 size, respectively. We can first expand all 3×3 kernels to 5×5 (see Figure 11b), merge two 5×256 conv operators into one 5×512 conv operator, and then separate them using *split* operator before executing the *concat* operator (see Fig. 11c). Via computation graph substitution, we remove the computational-intensive conv operator. And the computational cost of the *split* operator is almost negligible.

Existing DL frameworks (*e.g.*, PyTorch [176], TensorFlow [82]) substitute computation graphs using greedy rules or manual methods, which, however, cannot guarantee the selection or combination bring rigorous improvements. Jia *et al.* [124] proposed an optimizer for graph substitution.

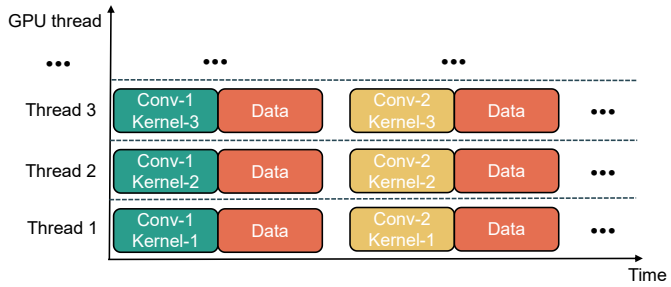


Fig. 12: Illustration of intra-operator parallelism technique. The conv operator kernels and data are divided into different groups and deployed on different threads to execute in parallel.

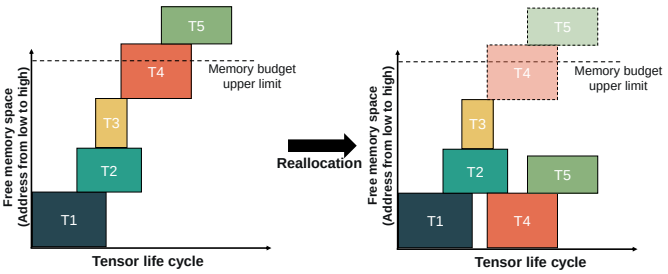
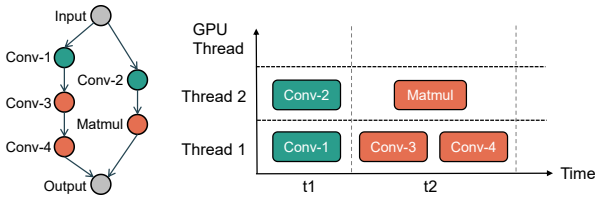


Fig. 14: Illustration of memory allocation technique. After reallocating the memory position of tensors, the memory peak is reduced without conflicting access to the tensor.



(a) Computation graph (b) Inter-operator parallelism

Fig. 13: Illustration of inter-operator parallelism technique. (a) the original computation graph, (b) two conv operators execute in parallel on different threads during the t_1 time period. During the t_2 time period, the *matmul* operator executes on thread 2, while the serial execution of two *conv* operators execute on thread 1.

They automatically use a cost-based search algorithm on the graph substitution space to find the optimal solution. Jia *et al.* [183] proposes an optimizer (*i.e.*, TASO) to substitute the computation graph automatically. TASO generates several candidates for a given list of operators and picks the most suitable substitutions. Experiments show that TASO outperforms existing DL frameworks by 2.8× and significantly reduces human labor. Fang *et al.* [175] formally defined the computation graph substitution problem (*i.e.*, OCGGS) and narrowed the search space to sample the best solution.

c. Operator parallelism. There are two kinds of operator parallelism techniques, *i.e.*, intra-operator and inter-operator parallelism. Mainstream AIoT platforms always equip with multi-core CPUs and multi-core GPUs. Considering the increasing speed gap of diverse processors (*e.g.*, CPU and GPU), separate methods are designed for the CPU and GPU to overcome the memory access bottleneck. For CPU devices, the cache hides delays in accessing memory to reduce the pressure on memory bandwidth. GPUs do not use or only use relatively small caches, mainly through the parallelism of threads, to hide the memory access delay. When some threads are stuck due to memory access, another part of the threads will continue to execute and will not let processing units idle.

Intra-operator parallelism. Existing DL frameworks (*e.g.*, TensorFlow [82], PyTorch [176]) can support intra-operator parallelism, *i.e.*, parallelizing arithmetic operations (*e.g.*, convolution) within a single operator. In the convolution operator, kernels that will be executed on the same thread are divided into one group, and then the corresponding data is also grouped. Then, different groups of conv kernels and data are deployed on diverse threads for parallel execution, as shown in Figure 12. However, as high-performance hardware evolves, intra-operator parallelism is no longer efficient enough.

Inter-operator parallelism. Inter-operator parallelism allows multiple operators to execute on different threads in parallel, as shown in Figure 13. Ding *et al.* [75] proposed an inter-operator scheduler(*i.e.*, IOS). IOS can search highly optimized parallel schemes using a dynamic programming algorithm. And this approach can be generalized to existing DL frameworks. Experimental results show that IOS increases the inference speed by $1.1\times \sim 1.5\times$.

d. Memory allocation DL model parameters and intermediate activations take up large memory resources during forward propagation as models become more complex. The default memory allocation schemes by the operating system (*i.e.*, OS) always result in memory fragmentations, which cannot be used for other tasks. To solve this problem, memory reallocation is necessary. During DL inference, input tensors, weights, and intermediate activations are all one-offs, we can release them from memory in time after use. Also, we can allocate the same memory to different tensors that do not coincide with the usage time. As shown in Figure 14, T1 $\sim$ T5 represent a set of tensors. If we allocate memory for each tensor in order, it will easily reach the memory limit, like tensors T4 and T5. And since the lifetime of tensor T4 is completely disjoint with T1, they can be allocated in the same memory block. Similarly, T5 and T2 can also share memory.

Sekiyama *et al.* [184] proposed a profiler-guided memory allocation method. During propagation, they collected information about requested memory blocks and then allocated memory using a heuristic algorithm to reduce peak memory footprint. By experimenting on advanced DL models, they reduced memory footprint by nearly 50% and improved inference speed by 4×. To consider the layer diversity of computation and communication, Wei *et al.* [177] designed a layer-wise memory allocation framework on Field Programmable Gate Array(*i.e.*, field-programmable gate array(FPGA)). They used the layer diversity and the non-overlapping lifespan information of memory buffers to schedule on-chip memory.

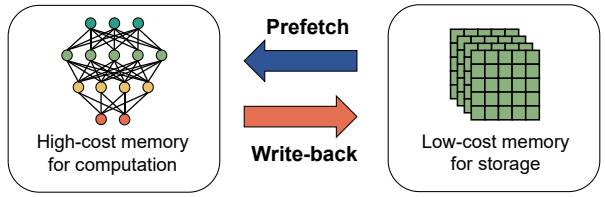
TABLE I: Summary of model-adaptive system scheduling-level enabling techniques for on-device DL inference.

Category			Technique highlight for improving	Device	Year	Ref.	Compiler frontend	Compiler backend
Model-adaptive system scheduling level	Computation graph level	Operator fusion	Circular fusion, operator classification, reduce redundancy, reduce computation latency	Mobile phone	2021	[77]	✓	
			Directed acyclic graphs, accurate memory cost model, reduce computation latency	Cloud server	2022	[76]		✓
			Candidate exploration, selection of fusion plans, code generation of local and distributed operations, reduce computation latency	MPU	2018	[81]	✓	
		Computation graph substitution	Relaxed graph substitution, backtracking search algorithm, flow-based graph split algorithm, reduce computation latency	Cloud server	2019	[124]		✓
			The first DL computation graph optimizer, automated theorem prover, cost-based backtracking search, reduce computation latency	Cloud server	2019	[183]		✓
			Pruning-based algorithm, sampling heuristic, reduce computation latency and memory usage	Cloud server	2020	[175]	✓	
		Operator parallelism	Intra-operator parallelism, reduce computation latency	Cloud server	2016	[82]		✓
			Intra-operator parallelism, reduce computation latency	Cloud server	2019	[176]		✓
			Inter-operator parallelism, inter-operator scheduler, novel dynamic programming algorithm, reduce computation latency	Cloud server	2021	[75]		✓
	Resource scheduling level	Memory allocation	Novel profile-guided memory optimization, heuristic algorithm, reduce computation latency and memory usage	Cloud server	2018	[184]		✓
			Memory allocation algorithm, memory bound layers, reduce computation latency and memory usage	FPGA	2019	[177]		✓
			Loop-orderbased memory allocation, fast auto-scheduling methodology, reduce computation latency and memory usage	Cloud server	2021	[178]		✓
		Memory swapping	The first memory swapping on MCU, swap data block between SRAM and FLASH or SD card, reduce memory usage	MCU	2021	[78]		✓
			Joint optimization along 3 dimensions, custom-designed genetic algorithm, reduce computation latency and memory usage	Cloud server	2020	[185]		✓
			Swap model parameters from the external storage into DRAM, task bounded with subnet, reduce computation latency	Cloud server	2022	[179]		✓
	Hardware instruction optimization	Loop unrolling	A generalized loop-unrolling method for any type of loop construct, reduce computation latency	Cloud server	1999	[79]		✓
			A semi-automatic and compile-time approach for identifying the optimal unroll factors, reduce computation latency	Cloud server	2010	[180]		✓
		Register blocking	A performance model to set the appropriate register block size, reduce computation latency	MPU	2001	[80]		✓
		Instruction reordering	A flexible multi-criteria instruction reordering heuristic that can be adapted across architectures, reduce computation latency	Cloud server	2018	[186]		✓

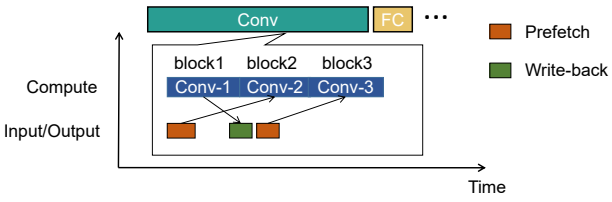
The hit rate on the hardware accelerator and the memory allocation policy mapping can also affect its performance. To search for the best map fastly, many works redesigned the search space, *e.g.*, state-of-the-art [187]–[190]. However, the efficiency in these DL frameworks is still slow due to exhaustive searches. And it does not consider the user-defined or random-sampled constraints, thereby cannot guarantee the global optimum. Also, they cannot deterministically assess the optimality because predicting the needed CPU time and peak memory in advance is prohibitive. To address this problem, Symons *et al.* [178] allocated memory based on the loop order. It takes advantage of DL models' nested "FOR" loop sequence and assigns them to the most appropriate memory hierarchy.

e. Memory swapping. Memory swapping [185], [191] refers to the exchange of tensors or data chunks between the high-cost memory and the low-cost one during computation. For example, when the device's memory supply cannot meet the tensors' demands, we can swap out partial tensors to free up space and serve current operations. The focus of memory swapping in the inference phase differs from that in the training phase. The training phase focuses more on the reuse of tensors during backpropagation. While the inference mainly focuses on forward propagation, *e.g.*, prefetching of tensors and the write-back after the computation is completed on memory-scarce AIoT devices, as shown in Figure 15a. However, memory swapping brings transmission delay. To solve this problem, some works propose to cover the transmission delay with the computation delay so that the transmission delay does not affect the overall delay. When the device resources are extremely limited, the model can be divided into several blocks for execution. For example, as shown in Figure 15b, when block 1 is executed, the parameters required by block 2 are prefetched, and the result is written back to the low-cost memory after block 1 is executed.

Miao *et al.* [78] proposed a system solution for deploying DL models on MCUs. It dynamically swapped model blocks between the SRAM and flash of MCU. This method trades time for space, thus not affecting accuracy. Besides, swapping



(a) Memory swapping process.



(b) Memory swapping delay.

Fig. 15: Illustration of memory swapping techniques for on-device DL inference.

tensors between GPU and CPU is also efficient because the GPU is fast with small memory. In contrast, the CPU has relatively large memory to store temporary tensors. And this method becomes more promising with the development of current GPUs. They can realize cross-communication and computation based on generous communication bandwidth. Huang *et al.* [185] proposed a universal swapping system called SwapAdvisor. It establishes search space with memory allocation and operator scheduling techniques and uses a genetic algorithm to determine exactly when and which tensors to swap before execution. SwapAdvisor breaks through the GPU memory limit by 12× and increases the inference speed by 4×. Ji *et al.* [179] proposed task-aware swapping(*i.e.*, TAS) for object detection tasks on IoT devices. Since the same type of task involves the same subnetwork, TAS swapped the model parameters of the corresponding subnetwork from external memory to dynamic random access memory(*i.e.*, DRAM) in time, according to different task types. TAS reduced the DRAM memory by 34.6% while maintaining accuracy.

f. Hardware instruction optimization. DL inference optimization on AIoT devices also includes the following hardware instruction optimization techniques:

Loop unrolling. It is a widely known code conversion method to improve program execution performance. The unrolled loop typically executes fewer instructions than the original one. As shown in Figure 16, the assembly loop code on the right is unrolled, and the calculation is reduced from 404 to 229 times. Thus loop unrolling can speed up calculations. Huang *et al.* [79] proposes a generic loop unrolling approach for diverse loop structures (*e.g.*, FOR, WHILE, DO-WHILE). The loop unrolling support in the GPU compiler is limited. Giridhar [180] developed a semi-automatic compiler to identify the optimal unrolling factor based on compile-time characteristics and the effect of loop unrolling on the program.

Register blocking. It rationally blocks registers to reduce idleness and increase register multiplexing. Based on matrix-independent device features, Sparsity [80] proposes a performance model to set the appropriate register block size, hence optimizing the sparse matrix computation speed.

Instruction reordering. It scrambles the instructions of different execution units to improve the utilization rate of the pipeline. Rawat *et al.* [186] proposed a flexible multi-criteria heuristic based on instruction reordering, which can be adapted across architectures.This approach alleviates register pressure while properly controlling the degree of parallelism at the instruction level.

The compiler or CPU/GPU/TPU processor can reorder instructions to optimize the execution performance of the program. For example, instructions with high delay are advanced, and data dependence before and after instructions is reduced. From the source program to the final running instructions, there are two stages of reordering:

Compiler reordering. During compilation, without affecting the result of the program, the compiler reorders instructions based on context analysis to reduce the interaction between CPU and memory. After rearranging, the CPU can read data from registers or cache rows as much as possible.

Processor reordering. It includes *parallel instruction set reordering* and *memory system reordering*. First, in the *parallel instruction set reordering*, modern processors use Instruction-Level Parallelism (ILP) to execute multiple instructions. The processor changes the order in which a statement corresponds to a device's instruction. As shown in Figure 17, the CPU hopes to execute the *instruction 1* and *instruction 2* in parallel. However, both them assign value to the same register *eax*, so the value saved in the final register *eax* is not the result of two addition computations. As a result, the value saved in register *ebx* is incorrect. Therefore, *instruction 1*, *instruction 2*, and *instruction 3* are executed serially. At the same time, the CPU needs to select one of the subsequent instructions (*i.e.*, *instruction 4*) to execute with *instruction 1* in parallel. With the reordered instruction set, parallel execution can be more efficient. Second, as for the *memory system reordering*, since the processor uses cache and read/write buffers, loading and storing operations is performed out of order. Therefore, memory system reordering should be well-designed.

3) Intra-device cross-level controller: The controller is required upon the above optimization techniques to automatically adapt to diverse performance requirements of AIoT applications and resource budgets of AIoT devices. Liu *et al.* proposed AdaDeep [74], which leverages the deep reinforcement learning-based optimizer to automatically select the most appropriate combination of compression techniques and hyperparameters for a given DL model in a hierarchical manner. Yang *et al.* [192] proposed an energy-aware CNN pruning algorithm, which automatically guides the pruning process by using the energy estimation of DL model. Edgar *et al.* [71] automatically design suitable DL models for MCUs, achieving high accuracy while achieving low memory usage and latency. MCUNet [12] is an example of cross-level system optimization, which can drive better optimization strategies. They jointly optimize the DL algorithms by TinyNAS and the memory scheduling by TinyEngine to reduce memory usage. TinyNAS automatically optimizes the search space

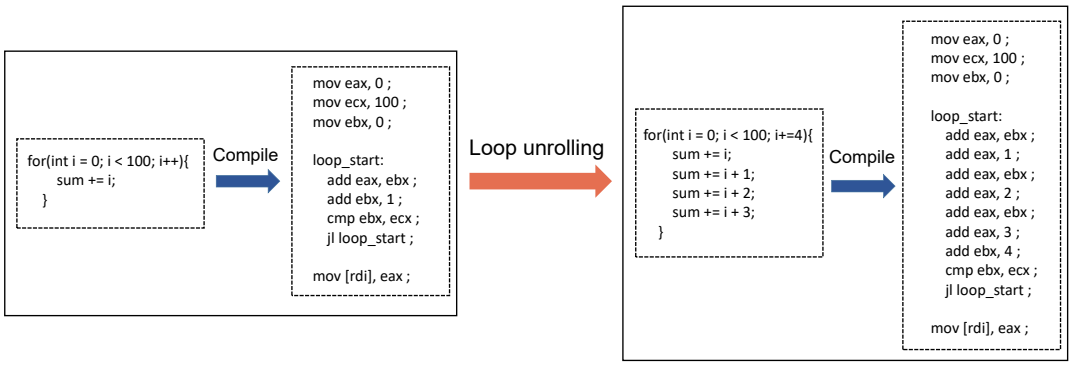


Fig. 16: An example of loop unrolling technique.

TABLE II: Summary of related intra-device controllers for on-device DL inference.

Category	Context awareness	Controller	Cross-level	Optimizated performance	Year	Ref.
Controller	Latency, memory	DQN, DDPG	Algorithm level	Accuracy, energy consumption	2020	[74]
	Accuracy	/	Algorithm level	Energy consumption	2017	[192]
	Peak memory usage, model size, latency	Multiobjective constrained NAS algorithm	Algorithm level	Accuracy, peak memory usage, model size, latency	2021	[71]
	Latency, energy, memory	Two-stage NAS for tiny memory constraints	Algorithm level & compiler level	Latency, memory	2020	[12]

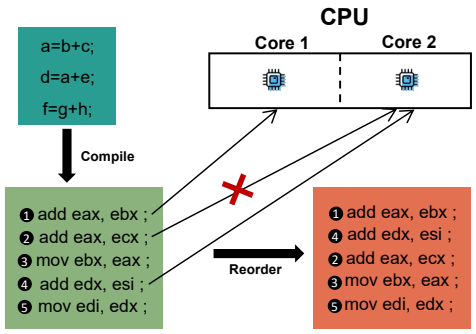


Fig. 17: Illustration of parallel instruction set reordering technique. The digits $1 \sim 5$ represent the instruction ID, not the actual execution order.

to fit the tiny resource constraints in MCUs. TinyEngine improves the existing inference library with code generator-based compilation methods to eliminate memory overhead. MCUNet has improved inference speed by 3.4× and reduced the peak memory footprint on SRAM by 4.1 ×.

B. Distributed DL Inference

In addition to on-device optimizing to reduce the resource demands of DL models for local adaptation, distributed DL inference aims to aggregate more resources from multiple devices within the networked AIoT systems to improve inference efficiency. This is achieved by partitioning the intensive computations of DL inference tasks and offloading diver portions of them to multiple devices. Distributed DL inference is particularly beneficial given the increasingly complex structure of modern DL models, which often require memory and computing resources far beyond the limits of a single AIoT device. For example, MCUs typically have only 256kb of memory, while ResNet, a widely-used model, requires 7.2MB for parameter storage.

Given the challenges mentioned in II-B, we divide distributed DL inference optimization into the resource-friendly algorithm level, model-adaptive system scheduling level, and inter-device controller, as shown in Figure 18. The algorithm and system scheduling levels focus on improving the resource efficiency of given heterogeneous hardware from different aspects. The inter-device controller automatically selects devices and cross-level techniques according to dynamic AIoT context.

1) Resource-friendly algorithm level: It mainly specifies the model partition and offloading schemes for satisfying distributed inference performance demands and resource budgets. We classify them as *layer-wise* and *operator-wise* categories. Specifically, the layer-wise scheme refers to partitioning and offloading the model according to model layers. And the operator-wise scheme goes deep inside layers, *e.g.*, optimizes the operator itself or the connection between operators.

Layer-wise scheme. Several studies [84], [196]–[198] have put forward the layer-wise distributed DL inference. Yun *et al.* [84] specialized the lightweight models by knowledge distillation and selected model partition points to minimize inference delay and satisfy the resource limitations of end devices (*e.g.*, Raspberry Pi 3B). Wu *et al.* [85] expressed the data sampling rate and model offloading problem as a constrained Markov decision process and obtained a heuristic solution. He *et al.* [86] model the arrival process of DL inference tasks as Poisson distribution. And they established a multi-faceted evaluation method [199], [200] to profile the inference latency, accuracy, memory usage, *etc.* DeeperThings [90] established the joint optimization of model partition and device selection

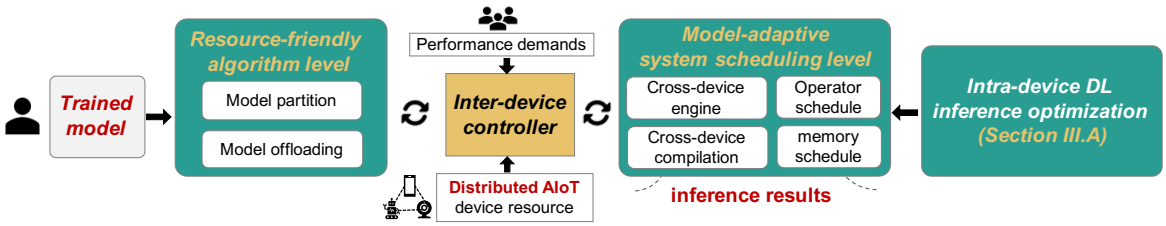
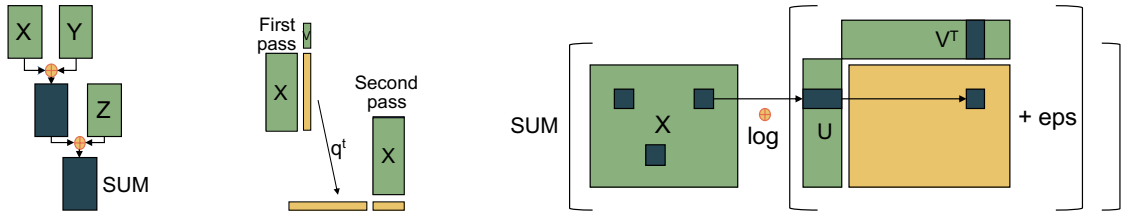


Fig. 18: System loop of the algorithm, system scheduling, and inter-device controller for distributed DL inference.



(a) Intermediates (b) Single-pass (c) Sparsity exploitation

Fig. 19: Illustration of code fusion techniques. (a) Code fusion can eliminate the intermediate in the computing, $sum(X \oplus Y \oplus Z)$; (b) code fusion can eliminate unnecessary scans of inputs, $X^T(Xv) \rightarrow ((Xv)^T X)^T$; (c) code fusion allows sparsity exploitation across chains of operations, $sum(X \oplus \log(UV^T + eps))$.

TABLE III: Summary of cross-level optimization techniques for distributed DL inference.

Focus level	Technique highlight for improving resource efficiency	Offloading	Year	Ref.
DL model	Multi-dimensional resource management, deep reinforcement learning	Serial	2021	[91]
	Constrain Markov decision process, Lyapunov optimization	Serial	2020	[85]
	Single batch inference, use several new model parallel methods	Parallel	2020	[93]
Operator	Operator fusion, distributed code generation	Hybird	2018	[81]
	Scalable convolution layer fusion, improve data reuse	Parallel	2018	[89]
	Partition computing layer, integer linear programming	Parallel	2021	[90]
	Convolution layer fusion and tiling, memory usage predictor	Hybird	2021	[193]
	Partition the full connection layer, cover all layers	Parallel	2019	[194]
	Vertical partition, weight pruning technique	Parallel	2022	[195]
Computation graph	Memory-constrained, joint optimization, knowledge distillation	Serial	2022	[84]
	Delay-Sensitive, mixed integer nonlinear programming,mec server	Hybird	2020	[86]
	Distributed model computing system, dynamic partition	Parallel	2020	[92]
	Reduce non-parallel data transfer time, convolution segmentation	Hybird	2017	[87]
Inter-device controller	Supports customized flexible fine-grained scheduling	Parallel	2021	[94]
	Greedy two-dimensional partition, structured models are tightly deployed	Hybird	2017	[88]

as a nonlinear programming problem. He *et al.* [86] designed a CRA algorithm based on Markov approximation to search the solution quickly, *e.g.*, 350 ms for a 10-device cluster.

Operator-wise scheme. Most studies focus on integrating and reallocating operators to reduce memory footprint. Specifically, merging more operators helps reduce the memory footprint of intermediate outputs and achieve better cross-operator sparsity development. For example, Boehe *et al.* [81] reduced the intermediate activation of the fusion of different operators and designed a cross-operator sparsity plan to improve computational efficiency. Stahl *et al.* [194] conducted research on layer operator fusion and reduced the data transmission time between multiple devices by combining feature and weight division with communication-aware layer fusion methods. Further, Farley *et al.* [193] design an independent fusion scheme for conv layer and fc layers, which reduces memory overhead through data reuse. However, these works lead to high synchronization overhead. To this end, Zhang *et al.* [94] presented an adaptive cooperative inference system that supports mainstream models (*e.g.*, ResNet, GoogleNet). And it provides a set of APIs to obtain the data location for enabling fine-grained scheduling and memory reclamation.

2) Model-adaptive system scheduling level: This level aims to optimize the input data reuse, intermediate data movement, and memory overhead (*e.g.*, memory fragmentation and recycle) in distributed DL inference [87]–[90], [201], [202]. It should be optimized separately at each device and verified globally cross-device. And how DL models offload to multiple AIoT devices affects the separate system scheduling technique selection and global performance verification. The existing

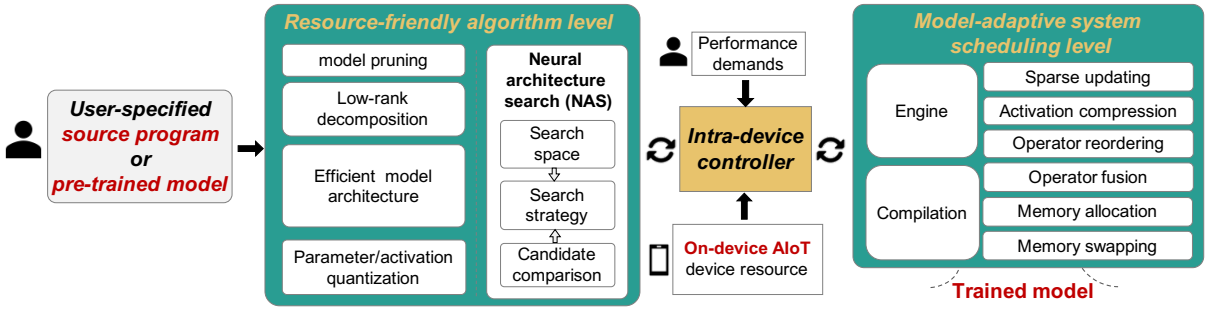


Fig. 20: System loop of the algorithm, system scheduling, and intra-device controller for on-device DL training.

DL model offloading schemes can be serial [84], [85], [91], parallel [89], [90], and hybrid [88], [193], [203]. For example, MoDNN [87] partitions the conv layer and distributes them to diverse AIoT devices for parallel computing. It partitions the input of the conv layer according to the matrix size and balanced unloading and sends them to different devices for collaborative inference. MeDNN [88] is an improvement of MoDNN, solving the problem of unbalanced model partition. It proposes a greedy 2D model segmentation algorithm to perform static load balancing according to the computing power of each device. Further to Mednn, Deepthing [89] proposed the improved solution. It proposed a scalable method to merge and partition the convolution layer by dividing the input matrix into different areas and unloading it to different devices for inference. It implemented a distributed job-stealing approach to realize dynamic workload allocation and computational efficiency balance, improving data reuse by 68% and reducing latency by $\geq$ 1.7 times. Another study [193] also explores the fusion and optimization of CNNs. Besides, it is difficult for existing operator fusion heuristics to develop distributed operator fusion schemes for complex models, *e.g.*, DAGs. Matthias *et al.* [81] proposed an optimization framework for systematic inference fusion schemes, Figure 19 shows three main ways of code fusion.

3) Inter-device cross-level controller: The context-aware controller can automatically adapt to varying contexts without re-designing systems, which is necessary for long-term running applications. Adjusting the distributed inference configurations for dynamic demands and heterogeneous devices is one of the recent research focuses [93], [94], [182], [204]. Haddi *et al.* [93] integrates multiple existing parallelism inference technologies into an automated framework. Zhang *et al.* [94] proposed Deepslicing, an adaptive control system integrating multipliers and deep reinforcement learning. It also exposes a set of APIs to users for adaptation. However, the controller across several AIoT system levels, especially the underlying system scheduling level, is still lacking.

To be compatible with heterogeneous AIoT devices and boost the inference efficiency, prior efforts also present the inter-device schedule frameworks, such as [92], [205]–[207], adapting to heterogeneous AIoT platforms/clusters. For example, Zeng *et al.* [92] presents Coedge. It involves a distributed DL inference framework and a fast approximate solution to determine the optimal platform scheduling strategies. Zhang *et al.* [91] considers the embedded chips' memory limits, computing frequency, and battery and transforms the hybrid optimization problem into a Markov decision process. As a result, it reduces inference delay and pushes the average accuracy limit caused by pre-determined resource allocation.

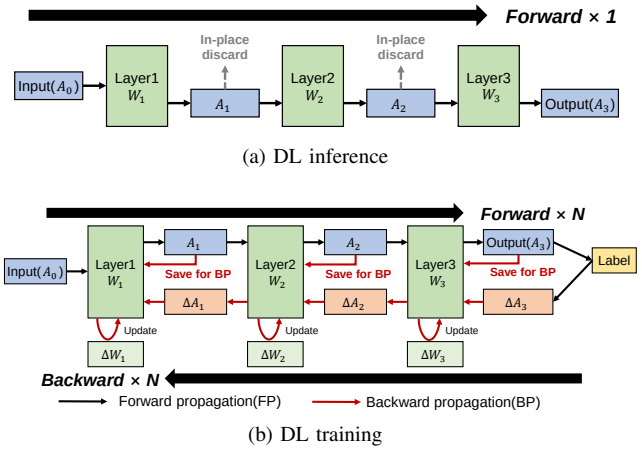
Discussion. Table I summarizes the standalone system scheduling techniques. And Table III illustrates the enabling techniques for distributed DL inference optimization across diverse levels. The performance of different levels of optimization techniques on the same DL model varies. And even within the same level of optimization techniques, their performance also differs. Existing DL frameworks for AIoT devices already support the techniques mentioned in this table. However, the criterion for their cross-level combination in the context of AIoT applications need more exploration. Moreover, combining diverse techniques across these levels with an adaptive and automatic controller is still lacking.

IV. Cross-level Optimization for DL Training

DL training adopts *batched memory* chunks grouping multiple data samples. And the memory usage increases proportionately to the batch size. Specifically, the computation efficiency will be low if not secure sufficient batch size. Larger batch sizes can bring more accurate distribution statistics for operators like BatchNorm [208] to speed up the training convergence and increase the accuracy.

We first **differentiate the specificities of DL training** from DL inference in terms of memory and computation demands, as shown in Figure 21. *(i)* DL training requires more computation than inference. More memory access during the training process also brings more memory access delay. For example, the bottleneck of GPU computing power for the GPU devices always lies in the memory access bandwidth and the upgrade of successive generations of GPU focusing on memory bandwidth proves this point [112].

(ii) DL training requires more memory space than inference. According to the chain derivative rule of backpropagation, calculating the gradient and derivative of the weights in the i^{th} layer requires using the derivative of the $i+1^{th}$ layer and the input of this layer. Therefore, the *intermediate activations* A_1, A_2 produced during forward propagation need to be saved till backpropagation calculating the gradients ΔA, ΔB and updating the weights during backpropagation. In contrast, A_1, A_2 in model inference process do not need to be preserved.



(a) DL inference

(b) DL training

Fig. 21: Comparison illustration of DL training and inference. (a) Inference only requires once forward propagation(FP), and intermediate activation tensor Ai can be discarded to reduce memory usage. (b) In training, tensor Ai needs to be saved for back propagation after the forward propagation.

They can be released right after they have been used, which brings a bigger memory requirement to the training process.

(iii) DL training needs N rounds of loop iteration, while inference needs only one round. For algorithm level (*e.g.*, model parameter/activation quantization), if not carefully designed, the tiny instabilities generated will be amplified in thousands of iterations and even leads to training crash [38]. Compared to the "one-time" property of the model inference process, DL training needs to consider optimization across *multiple iterations*, which also challenges system optimization.

A. On-device DL Training

It is non-trivial to optimize three key performance metrics, *i.e.*, *latency*, *memory*, and *accuracy*, simultaneously for on-device DL training. Given challenges towards optimizing memory usage, latency, and accuracy simultaneously (see § II-C), we summarize the on-device DL training optimization techniques into diverse system levels as described in § II-B, *i.e.*, the resource-friendly algorithm level, model-adaptive system scheduling level, and inter-device controller. Figure 20 shows their relationships in the system loop.

1) Resource-friendly algorithm level: We first briefly introduce the algorithm-level techniques, *i.e.*, compressing DL model structure [116] [209], reducing parameter/activation bit width [115] [210], and sparse updating [118] [119].

a. Parameter/activation quantization. DL model quantization during training is more complicated than inference since it brings instability to training [211] [212] [213]. Deng *et al.* [115] conducted the first exploration of model quantitation in DL training. They implemented reduced-precision memory access of parameters and saved significant memory bandwidth using an approximator. Subsequently, Zhou *et al.* [210] train DL models with low-bit width weights, activations, and gradients on diverse devices, *e.g.*, CPU, GPU, application specific integrated circuit(ASIC), and FPGA, to speed up training. Besides, Micikevicius *et al.* [214] maintain a single-precision copy of weights to prevent information loss caused by quantification. Thus they preserve gradient values with small magnitudes and result in half-precision arithmetic. Huang *et al.* [127] proposed the adaptive precision training (APT) method to balance energy cost, memory usage, and accuracy in DL training. Motivated by [215], they find that starting DL training with low precision benefits energy and memory savings. APT dynamically allocates layer-wise bit precision, allowing model to learn faster. Because once the training curve reaches a plateau, increasing precision allows DL training to approach better accuracy with fewer training epochs. And it uses an application-specific hyperparameter to balance the abovementioned three metrics automatically. Wang *et al.* [216] find that during the backpropagation in training, mainly the activation maps' low-frequency component (LFC) is used. As such, they preserve the high-precision copy of LFC while compressing the high-frequency component (HFC) into a low-precision copy. This greatly reduces memory cost while not dramatically decreasing the precision of backpropagations. To realize DL training quantization on MCUs lacking DL training frameworks and low-bit width APIs. Yu *et al.* [217] deploy sub-byte models on MCUs efficiently. They propose a training framework for low-precision models, followed by direct buffer convolution and packed sub-byte multiply-accumulation to accelerate on-device training. Lin *et al.* [38] proposed quantization-aware scaling to alleviate the gradient scale mismatch issues caused by mixed bit-precision training. They stabilized the quantized training by calibrating the gradient scales for MCUs. Instead of improving the quantizer functions, Lu *et al.* [218] optimize the training process from a weight-searching standpoint.

b. Efficient model architecture. DL model compression is another promising method to optimize the AIoT system performance for DL training. Bulo *et al.* [116] proposed in-place activated batch normalization (BN) that eliminates intermediate results and recovers required information during the backward pass, leading to a 50% reduction in memory usage. It can be easily applied on existing models (*e.g.*, ResNet [219], DenseNet [220]) with a minor increase (about 2%) in training latency. Besides, Jung *et al.* [117] split the BN layer into two sub-layers to restructure BN layers. Because existing memory access optimization approaches, such as fusing convolution layers, are ineffective for accelerating BN due to their inability to optimize mini-batch-related calculations during training. To significantly reduce main memory accesses and optimize calculations related to mini-batch, they combined the first sub-layer with the preceding conv layer and the second sub-layer with the following conv layer. To reduce memory usage, some studies also proposed new model structures. *e.g.*, tiny-transfer-learning (TinyTL) [112] added the lite residual module with low-dimensional features and group convolution to improve the model's migration ability. And it avoids intermediate activation storage by only training bias and lite residual modules. This approach leads to a reduction in training memory costs at near-constant precision. Yang *et al.* [209] proposed a reprogramming network, which trains the model using the new task data to reprogram the intermediate features of a backbone model, leading to lower training memory and higher accuracy.

c. Sparse updating. Like pruning in the inference phase, sparse updating does not change the model structure but selects a part of the network to update in each training epoch. Most pruning studies focus on DL inference and are unsuitable for DL training. Liu *et al.* [118] utilize dynamic and sparse graphs (DSGs) in DL training, activating a few neurons during each iteration, resulting in considerable memory savings with competitive accuracy. Dai *et al.* [119] proposed SparseTrain, which employs a stochastic pruning algorithm for each layer and a sparse architecture with 1-dimensional convolution dataflow to realize implicit and artificial sparsity for training acceleration. Lin *et al.* [38] skipped the gradient calculation of insignificantly necessary layers and tensors. And they achieved the near-optimal solution with sparse updating through an evolutionary search. Kaplan *et al.* [221] presented SubTuning, which only trains a carefully selected subset of layers depending on the fine-tuning profile while freezing the remaining. They proved that SubTuning fastly obtains training convergences and even outperforms full fine-tuning when training data is lacking.

2) Model-adaptive system scheduling level: It further maximizes system performance and resource efficiency beyond the optimization capabilities of the algorithm-level techniques and thus pushes the limit of performance-resource tradeoff. As shown in Table IV, it optimizes the intermediate activation tensors and operators in the computation graph, optimizes the runtime/compiler of the DL frameworks, and re-allocates memory/computing resources.

a. Recomputation. In DL training, the intermediate activations generated during the forward propagation are typically stored until the backpropagation to calculate the gradient, which causes a large memory footprint. Experiments have shown that model's intermediate activations use much more memory than parameters [112]. To this end, recomputation methods [120], [121], [223] discard part of intermediate activations right after forward computation to save memory usage and recalculates these activations during backpropagation for gradient calculation, as shown in Figure 22. Chen *et al.* [120] is the first to present the recomputation technique, which splits the model into several parts and keeps only the first activation of each part after forward computation. During backpropagation, activations within each part are computed from the first activation retained. This work realizes significant memory savings. And the recompense of computing delay for memory space is also beneficial, especially when sufficient computing power is available. Following it, Gruslys *et al.* [222] use dynamic programming to find a storage strategy that optimizes computational cost for a given memory budget. Gomez *et al.* [121] applied recomputation in ResNets to save memory footprint during backpropagation. These recomputation methods were mainly implemented on cloud servers with sufficient computing power. The recomputation operations may bring unacceptable additional latency overhead for devices such as Raspberry Pi and MCUs. [141], [224] apply recomputation strategies on mobile phones, using slight additional latency in exchange for significant memory savings.

b. Intermediate activation encoding. Similar to but different from recomputation, intermediate activation compression

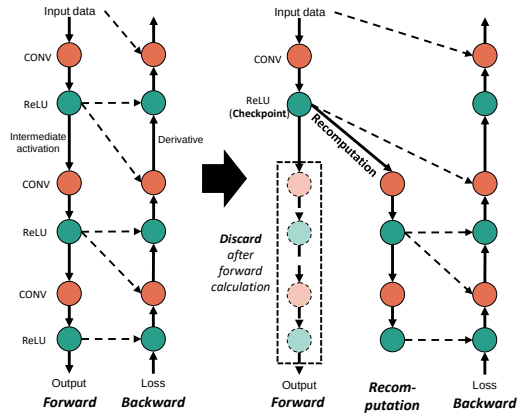


Fig. 22: Illustration of recomputation technique. The intermediate activation is discarded after forward computation and recalculated before backpropagation.

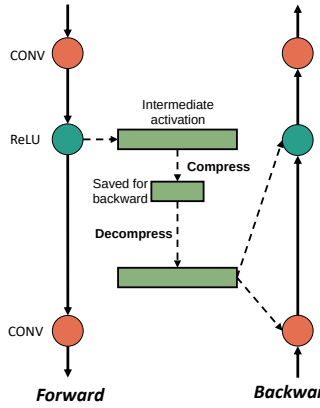


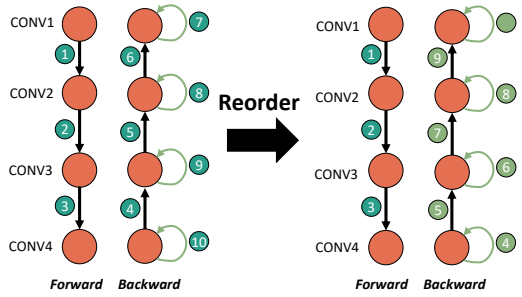
Fig. 23: Illustration of intermediate activation encoding technique. Intermediate activation is encoded after forward computation and decoded during backpropagation.

does not directly discard intermediate activations but *temporarily encode* activations after forward propagation and then *decode* them during backpropagation to calculate gradients, as shown in Figure 23. It thereby strikes a valuable balance between computing latency and memory savings rather than simply trading latency for memory resources. Specifically, Jain *et al.* [122] proposed Gist, a system that employs layer-specific encoding schemes, lossless and lossy, to significantly reduce the memory consumption of targeted feature maps. They store an encoded representation of feature maps and decode them for use in the backward pass; the full-fidelity feature maps are used in their forward pass and relinquished immediately. Evans *et al.* [225] proposed JPEG for activations (JPEGACT), a lossy activation offloading accelerator. They expand the well-known JPEG algorithm for 2D image encoding to compress model's activation. And recently, Hosny *et al.* [226] proposed BitTrain, a novel bitmap compression technique using activation sparsity to reduce the memory footprint during training.

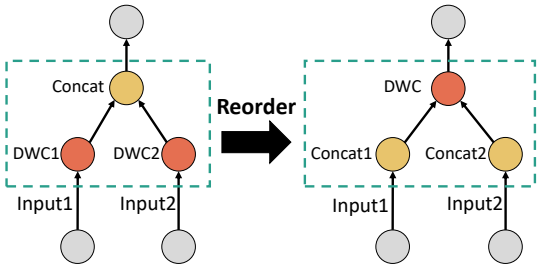
c. Operator reordering. The traditional DL training process retains every gradient in the backpropagation and updates weights uniformly after calculating all gradients. Re-ordering

TABLE IV: Summary of model-adaptive system scheduling techniques for on-device DL training.

Category				Highlight technique for improving resource efficiency	Year	Ref.	Compiler frontend	Compiler backend
Model-adaptive system scheduling level	Computation graph level	Intermediate activation	Recomputation	Split model and only retain part of activations, reduce memory usage with extra computation cost	2016	[120]		✓
				Use dynamic programming in recomputation, optimize computation cost	2016	[222]		✓
				Recomputation for ResNets, reduce memory usage with extra computation cost	2017	[121]	✓	
				Dynamic tensor rematerialization, reduce memory usage with less computation cost	2020	[223]		✓
				Memory-calibrated progressive recomputation on mobile phone, reduce memory usage	2022	[141]		✓
				Progressive recomputation based on memory worthiness of tensors, reduce memory usage	2022	[224]		✓
			Compression	Two Layer-specific encoding schemes, lossless and lossy, reduce memory usage with extra latency	2018	[122]	✓	
				Optimized JPEG method for activation compression and accelerator, reduce memory usage with extra latency	2020	[225]		✓
				A bitmap compression technique using activation sparsity, reduce memory usage with extra latency	2021	[226]	✓	
		Operator	Reordering	Discard gradients, reduce memory usage	2022	[38]	✓	
				Re-order operators to reduce the number of complex operators, reduce computation cost	2022	[227]	✓	
			Fusion	Relaxed graph substitutions by a backtracking search algorithm, reduce execution time and memory usage	2019	[124]		✓
				Transferable deep reinforcement learning for searching, reduce execution time	2020	[125]	✓	
				Account for memory access constraints, a unified memory function, reduce execution time	2020	[228]	✓	
				Merge memory-intensive operators into large GPU kernels, reduce execution time	2020	[229]		✓
				Reduce data movement in memory for Transformer, reduce execution time	2021	[230]		✓
				Consider multiple optimization objectives for memory-intensive computations, reduce latency	2022	[231]		✓
	Resource scheduling level	Memory allocation		Split tensor into micro-tensor to allocate and swap memory, reduce memory usage and latency	2019	[232]		✓
				Tensor life cycle computation and memory sharing, reduce memory usage	2022	[114]		✓
				Tensor-lifetime-aware memory allocation algorithm, reduce memory usage, energy cost, and latency	2022	[141]		✓
		Memory swapping		The first work on DL memory swapping between host(CPU) and device(GPU), reduce memory usage	2016	[39]		✓
				Dynamically select convolution operators and memory swapping strategy, reduce device memory usage	2018	[126]		✓
				Combine recomputation and memory swapping, reduce memory usage	2018	[40]		✓
				Use categorized topological ordering to simulate graph execution, reduce memory usage	2019	[233]		✓
				Decrease amount of synchronization operations and memory offload decisions, reduce memory usage	2019	[234]		✓
				Joint optimization on operator scheduling, memory allocation, and swap decisions, reduce memory usage	2020	[185]		✓
				Memory management based on dynamic tensor access pattern, reduce device memory usage	2020	[191]		✓
				Optimize tensor management on heterogeneous memory, reduce memory usage	2021	[235]		✓
				Selectively compress tensors before memory swapping, reduce memory usage	2021	[236]		✓
				Memory swapping without high-level information of computation graphs, reduce memory usage	2022	[237]		✓
				Improve prefetching using correlation tables to speed up memory swapping, reduce memory usage	2023	[238]		✓



(a) By modifying the operator's execution order, the gradient of each layer is discarded immediately after the weight is updated.



(b) Reording operator can also decrease the number of computation-intensive operations.

Fig. 24: Illustration of operator reordering technique.

operators within the computation graph during backpropagation can reduce peak memory usage [38]. Lin *et al.* [38] proposed the Tiny Training Engine (TTE) to reorder operators in DL training. As shown in figure 24a, by modifying the order of operator execution(exchanging gradient computation and updating operations) in back-propagation, TTE discards the i^{th} gradient immediately after updating the i^{th} layer instead of keeping it throughout the whole training iteration to achieve memory saving. Immediately freeing up space occupied by useless tensors benefits memory savings. Besides, Unity [227] not only changes the operator order but also reduces the number of computation-intensive operations. As shown in figure 24b, Unity exchanges the order of two parallel *Depthwise Conv* (DWC) operators and a *concat* operator to reduce the computation of the convolution process by reducing one DWC operator.

d. Operator fusion. Operator fusion in DL training focuses on merging multiple operators into one to boost computation and memory access efficiency. Suppose two adjacent layers (i^{th} and $i+1^{th}$ layer) in the computational graph are merged, the intermediate activation generated by the i^{th} layer can be directly applied to the $i+1^{th}$ layer without additional read and write transactions with the memory. As a separate note, the *computation graph optimization* that *operation fusion* techniques belong to is a broad field. Various operator fusion techniques have been well explored in existing research [11], [239] and DL frameworks, *e.g.*, TensorFlow [82], PyTorch [176], NCNN [240], and MindSpore [241]. Since fixed rule-based operator fusion cannot guarantee that the performance (*e.g.*, latency) is always optimal [9], [82], [83], MetaFlow [124] realized relaxed graph substitutions for operator fusion via a backtracking search algorithm to address this issue. Zhou *et al.* [125] proposed a transferable deep reinforcement learning-based optimizer to search for optimization policies to improve the optimization efficiency further. It speeds up the search drastically by making decisions based on the entire graph rather than on each node individually compared to previous techniques such as Hierarchical Device Placement(HDP) [242], Spotlight [243], and Placeto [244].

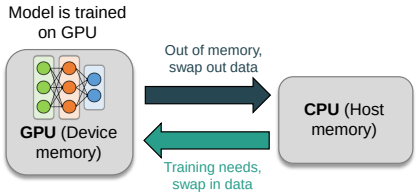
The operator fusion for computation graph optimization can also derive novel DL *compilers* or *frameworks*. For example, Zheng *et al.* [229] introduced a DL training compiler. It merges memory-intensive operators with data dependencies and non-homogeneous parallelism into large GPU kernels, reducing global memory access and context switch overhead. Hu *et al.* [228] proposed Jittor, a just-in-time (JIT) compiled DL framework. It integrated with enhanced operator fusion rules which accounts for memory access constraints as a unified function, allowing for unified management of the GPU memory. Ivanov *et al.* [230] reduced data movement for the Transformer by constructing a dataflow graph, identifying opportunities for data reuse and applying tailored fusion rules. Zheng *et al.* [231] proposed AStitch, a compiler that explores the new operator-stitching optimization space for memory-intensive computations.

e. Memory allocation. Memory allocation methods are typically implemented in the compiler backend, which is closer to hardware resources than the aforementioned techniques for optimizing computation graphs. In DL training, the time between the creation and final access of a tensor is referred to as the tensor's "life cycle." Existing DL frameworks divide a separate memory space for each tensor in the training time, much larger than most tensors' life cycles. Actually, it is useless to keep memory space for the tensor outside its life cycle [114]. Recycling memory for tensors by allocating the same memory for two tensors whose life cycles do not intersect can save memory usage, as shown in Figure 14. Moon *et al.* [114] analyzed all tensors (including input data, weights, and intermediate activations) that may appear during the training time and realized memory recycling, greatly reducing memory usage. Wang *et al.* [141] identified the memory allocation optimization problem is a 2DSP-like NP-hard problem [245]. Since tensors' life cycles can significantly impact layout effectiveness, they positioned long-lifecycle tensors beneath short-lifecycle ones to approximate the ideal solution to this problem. Nie *et al.* [232] presented TSPLIT, a fine-grained model memory allocation method that overcomes memory constraints while retaining DL training efficiency. With the tensor-splitting primitive, TSPLIT breaks the operation boundary of a tensor, allowing flexible memory allocation.

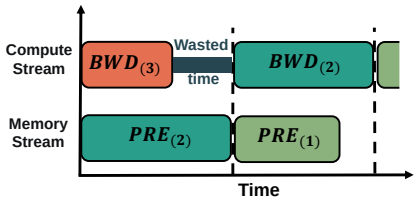
f. Memory swapping. In DL training, the intermediate activation of a certain model layer is *temporarily used* in the layer's forward and backpropagation. Swapping temporarily unused intermediate activations from the precious memory (*e.g.*, GPU) to the larger memory (*e.g.*, CPU) and then swapping back when gradient updating needs them can speed up computation and reduce memory usage, as shown in Figure 25a. Memory swapping is commonly supported by *operating systems*, *e.g.*, virtual address spaces in Windows [246] and

TABLE V: Summary of related intra-device controllers for on-device DL training.

Category	Focus level	Context awareness	Controller	Optimized performance	Year	Ref.
Controller level	Resource-friendly algorithm level including quantization	Gradient, energy, memory	/	Layer-wise quantified precision	2020	[127]
	Model-adaptive system scheduling level including checkpoints in computation graph	Memory budget	Dynamic programming	Total computational cost	2016	[222]
	Model-adaptive system scheduling level including checkpoints in computation graph	Tensor staleness, memory budget	Greedy, heuristics	Total computational cost	2020	[223]
	Model-adaptive system scheduling level including memory swapping	Memory budget	Genetic algorithm	Execution time	2020	[185]
	Model-adaptive system scheduling level including memory swapping	GPU architecture	Bayesian optimization	(De)Compression time	2021	[236]
	Co-design: resource-friendly algorithm level including quantization, sparse updating & model-adaptive system scheduling level including operator reordering	Memory budget	Evolutionary search	Accuracy	2022	[38]



(a) Memory swapping process.



(b) Memory swapping delay.

Fig. 25: Illustration of the memory swapping technique. $BWD_{(n)}$ and $PRE_{(n)}$ are the backward and prefetch computations for $layer_{(n)}$, respectively.

swaps partitions in Linux [247]. When the memory capacity of a device is insufficient, data will temporarily swap out to external memory like disks. However, it brings extra transfer delay. And memory usage is difficult to predict for random programs. Notably, this shortcoming can be well solved in DL training because the forward computation and backpropagation are in fixed orders (*i.e.*, forward computation from the first layer to the last layer and backpropagation is contrary), bringing optimization potential to memory swapping. For example, the data d^2 required by $layer_2$ backward propogation $BWD_{(2)}$ can be prefetched to GPU memory during $BWD_{(2)}$ computation. As a result, the d^2 transfer delay $PRE_{(2)}$ can be partially covered by $BWD_{(3)}$, or completely coverd like $PRE_{(3)}$, as shown in figure 25b.

Rhu *et al.* [39] proposed vDNN, the first work on memory swapping for DL training. They propose a runtime memory manager that virtualizes the memory utilization of models by enabling the concurrent usage of both GPU and CPU memory during training. vDNN reduces the average GPU memory usage of multiple mainstream models by over 90%. Following vDNN, Chen *et al.* [126] proposed moDNN, an intelligent solution capable of dynamically choosing convolution operators, adjusting mini-batch size, and selecting the suitable memory-swapping strategy to achieve optimal system performance. It realized fast Fourier transforms and Winograd with improved performance and increased memory requirements. Subsequently, Wang *et al.* [40] combined the memory swapping and recomputation methods and realized dynamic GPU memory scheduling that enabled DL training far beyond the GPU DRAM capacity. Le *et al.* [233] formally rewritten the computation graph and inserted swap-out and swap-in operations to store intermediate results on CPU memory. By introducing a categorized topological ordering to simulate computation graph execution, the memory consumption of a model can be profiled using operation distances in the ordering. Shriram *et al.* [234] proposed vDNN++ t o address the issues of delayed computation start, high pinned memory requirements, and GPU memory fragmentation in vDNN [39] by lowering the number of synchronization operations. Huang *et al.* [185] proposed SwapAdvisor to enhance the previously presented memory swapping methods with manual judgment. It jointly optimizes three dimensions of given dataflow graphs, *i.e.*, operator scheduling, memory allocation, and swap decisions. SwapAdvisor explores the wide search space through a genetic algorithm, improving the GPU's capability to accommodate large models, *e.g.*, WideResNet-152 [155], NasNet-25 [248], and BRNN-4-8K [249], they need 180GB, 193GB, and 99GB memory space for training, respectively. Following it, Peng *et al.* [191] conducted flexible memory management control by dynamically tracking the tensor access patterns at

runtime, which decide when and how to swap memory.

Furthermore, Ren *et al.* [235] presented Sentinel, a runtime system that automatically optimizes tensor management on heterogeneous memory. Specifically, it enables the optimal memory *co-allocation* for several tensors with similar lifetime and memory access frequencies by allocating them to the same pages to avoid unnecessary data movement. Then Chen *et al.* [236] optimize memory swapping time by selectively compressing tensors based on their sparsity, and size. Li *et al.* [237] automatically trace the memory behaviors of model workloads to schedule memory swapping without perceiving high-level information of layer structures or computation graphs, which alleviate the problem of existing solutions being densely coupled with the fixed model workloads, *e.g.*, layer structures or computational graphs. Jaehoon *et al.* [238] presented DeepUM, offering GPU memory oversubscription for models by leveraging compute unified device architecture (CUDA) unified memory and employing CPU memory as a backing store. They used a new correlation prefetching mechanism at UM block level to hide the fault-handling overhead. Thus it considerably reduces memory swapping time and increases GPU's virtual memory capacity.

3) Intra-device cross-level controller: The resource demands of DL training tasks vary. This variability, combined with the heterogeneous computing/memory resources of AIoT devices, can provide numerous optimization possibilities. For instance, when a device has tight memory and rich computing resources, it may be necessary to frequently employ recomputation techniques, while for a device with sufficient memory, such strategies can be avoided to reduce unnecessary computation delay. Furthermore, different training epochs on the same device can result in diverse performance outcomes. Therefore, an adaptive controller should be employed in the resource-efficient AIoT system to adjust the technique configuration based on the availability of resources and performance demands. A context-aware controller can prevent memory units from being idle or overloaded, thereby enhancing efficiency.

Table V summarizes existing adaptive controllers integrated with diverse levels of optimization techniques. For example, Huang *et al.* [127] proposed adaptive precision training in DL training quantization. It dynamically allocated layer-wise precision and provided an application-specific hyperparameter for users to achieve the trade-off between training energy cost, memory usage, and accuracy. In sparse updating, Lin *et al.* [38] adopted an evolutionary algorithm to search for the most important layers for updating and achieve higher training accuracy under a limited memory budget. In recomputation, Gruslys *et al.* [222] utilized dynamic programming to balance intermediate activations and recomputation caching. With a tunable memory budget, it can optimize computation costs. Kirisame *et al.* [223] extended recomputation methods by introducing dynamic tensor rematerialization (DTR). DTR is a greedy online algorithm that is parameterized by eviction policy. In memory swapping, Huang *et al.* [185] proposed SwapAdvisor, which uses a genetic algorithm to search for the optimal memory swapping scheme for reducing the transfer delay between CPU and GPU memory. Given the memory budgets, they used a dataflow engine simulator to quickly estimate the execution time of each scheme and find the best one with the lowest latency. Chen *et al.* [236] proposed a self-tuning tensor compression framework CSWAP. It used bayesian optimization to search for the optimal hyperparameters for tensor compression and tackle the heterogeneity caused by different GPU device architectures and DL frameworks.

B. Distributed DL Training

With the increase in DL model complexity and diversity, training models on AIoT devices at low cost is still challenging. To realize this intractable goal, distributed DL training on multiple AIoT devices is considered as the promising way [133], [250], [251]. This includes centralized systems such as model ensembling [252], decentralized systems such as tree-like topology [253] and parameter server [254], and fully distributed systems such as peer-to-peer [255]. Distributed DL training (with/without privacy concerns) partitions the data or DL models, reducing the memory load on each device compared to on-device training (as discussed in § IV-A).

Existing effort on distributed DL training (without privacy concern) mainly includes three categories, *i.e.*, model parallelism, data parallelism, and hybrid parallelism. Figure 27a shows their difference. When the DL model is too large to feed into a single device, it is necessary to adopt *model parallelism* to assign different parts of the model to diverse AIoT devices for training and finally merge them into a complete model. *Data parallelism* divides huge data into N parts to deploy N distributed AIoT devices without privacy concerns. Each AIoT device only processes $1/N$ of data and aggregates gradients to the central server for an update. *Hybrid parallelism* combines the strengths of the above two schemes. Federated learning is a widely known data parallelism method with privacy concerns. And in AIoT federated learning, the data on each AIoT device is sensed by itself and cannot be re-distributed. It only shares the model updates among multiple devices. Federated learning includes asynchronous, semi-synchronous, and synchronous schemes, as shown in Figure 27b. In this section, from the cross-level perspective of a resource-efficient AIoT system, we introduce enabling techniques at different levels, *i.e.*, resource-friendly algorithm, model-adaptive system scheduling, and inter-device controller upon them, illustrated in Figure 26.

1) Resource-friendly algorithm level: At the algorithm level, researchers explore partitioning the DL models for training parallelism or scheduling various devices in parallel to improve training efficiency. Federated learning is one of the widely-used distributed learning schemes. To reduce waiting time, Xie *et al.* [257] used asynchronous federated learning technology, adopting an asynchronous collaboration mode of upload-on-the-fly to minimize training time for each round. Then Ouyang*et al.* [259] proposed a novel federated learning system that can identify intrinsic similarities between data points, leading to higher model accuracy and lower communication overhead. In distributed learning without privacy concerns, Huang *et al.* [128] proposed GPipe, a scalable model parallelism library to boost the training efficiency of giant models, *e.g.*, generative pre-trained transformer (GPT) and bidirectional encoder representations from transformers

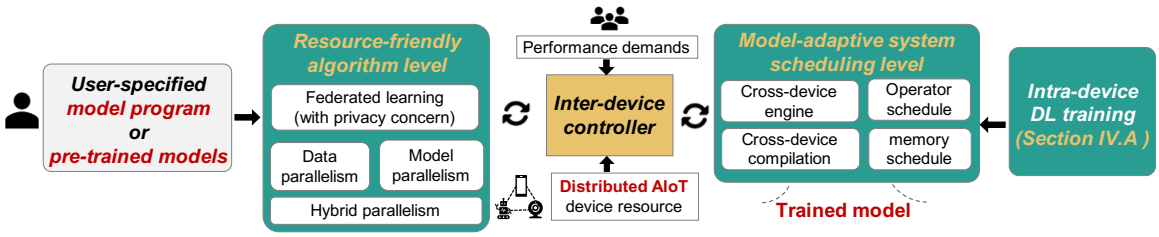
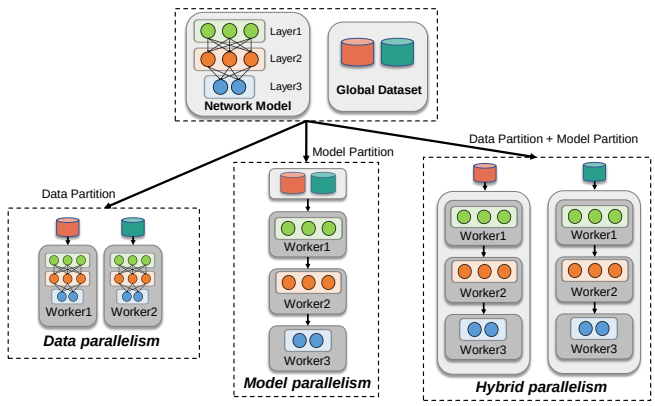
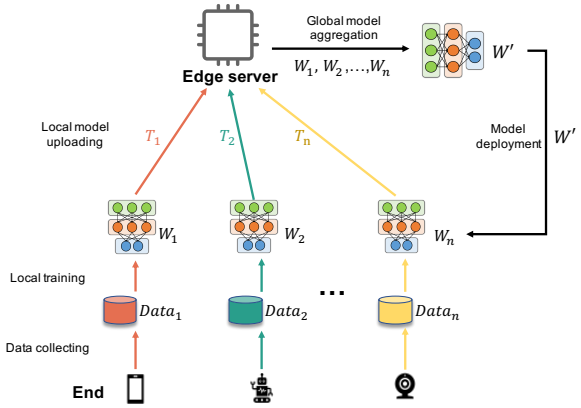


Fig. 26: System loop of the algorithm, system scheduling, and inter-device controller for distributed DL training.



(a) Distributed DL training (without privacy concern)



(b) Federated learning (with privacy concern)

Fig. 27: Illustration of different distributed DL training techniques. (a) Distributed learning (without privacy concerns) can be divided into data parallelism, model parallelism, and hybrid parallelism; (b) Considering different time points of model updating, federated learning includes asynchronous, semi-synchronous, and synchronous schemes.

(BERT). GPipe partitioned the model across different accelerators and split the mini-batch into smaller micro-batches for parallelism execution. By splitting batch, GPipe provided an almost linear speed up, *i.e.*, $3.5 \sim 20\times$, with no alterations to the model parameters. PipeDream-2BW [250] realized the memory-efficient DL training parallelism by utilizing a new weight gradient coalescing algorithm and weight double buffering. It efficiently split the DL model across available hardware resources considering hardware limits such as accelerator memory capacity and interconnect topologies. Beaumont *et al.* [256] proposed MadPipe to dramatically improve the training throughput by hybrid parallelism. It gives a more precise estimation of the memory requirements and a dynamic programming-based heuristic algorithm, which results in efficient computation-memory allocation and schedule.

2) Model-adaptive system scheduling level: It is intractable to enable the scalability and adaptivity of the distributed DL training to fully use varying resources in AIoT devices (*e.g.*, drone swarms and smart camera arrays) whose resources dynamically change due to the influence of other on-device tasks. Cui *et al.* [130] proposed GeePS, the first GPU-specialized *parameter server* design for realizing the data-parallel DL training on GPUs. Compared to previous CPU-based parameter server systems, GeePS employs GPU-friendly caching, data staging, and memory management techniques to reallocate GPU memory for intermediate layer state cache, significantly reducing training latency. Wahib [131] proposed a performance model based on the concurrent analysis of out-of-core training behavior and combined layer-wise memory swapping and redundant recomputing to decrease memory usage during training. Regarding scalability, the proposed out-of-core data parallel method outperforms complicated hybrid model parallelism in training huge models, *e.g.*, Megatron-LM [262] and Turning-NLG [263]. Rajbhandari *et al.* [260] proposed a zero redundancy optimizer (ZeRO) to optimize memory usage and improve distributed training speed while increasing the model size. ZeRO mainly employs two techniques, *i.e.*, reducing memory state redundancy across data-parallel processes by splitting the model states rather than repeating them; and proactively allocating memory based on the lifetime of various tensors to prevent memory fragmentation.

Md *et al.* [261] proposed DistGNN, which uses a minimum vertex-cut graph partitioning algorithm to reduce the transfer time of features and gradients. DistGNN further accelerates the shared memory system by using cache blocking, loop reordering, and vectorization with the LIBXSMM library [264], which considerably boosts distributed training throughput. Then Lim *et al.* [132] proposed Zico, the first model system which aims to reduce the memory consumption for parallelism training. Zico monitors the memory usage of each training task through its progress on GPU computation and reclaims the memory that is no longer needed, making it globally sharable. As a general approach, Zico outperforms existing GPU-sharing methods. Recently, Liu *et al.* [251] presented MAP, a PyTorch-

TABLE VI: Summary of cross-level optimization techniques for distributed DL training.

Category		Highlight technique for improving resource efficiency	Year	Ref.	Parallelism mode		
					Data	Model	Hybrid
Resource-friendly algorithm level	Distributed DL training (without privacy concern)	Compute layers in pipeline to reduce latency caused by distributed training	2019	[128]		✓	
		Pipelining and weight gradient coalescing to reduce memory usage and latency	2021	[250]			✓
		Dynamic programming based heuristic to estimate memory requirements more precisely, reduce latency	2022	[256]			✓
	Federated learning (with privacy concern)	Asynchronous federated learning to minimize waiting latency	2019	[257]	✓		
		Re-parameterization to decrease the convergence time	2020	[258]	✓		
		Similarity-aware federated learning system to reduce communication overhead	2021	[259]	✓		
Model-adaptive system scheduling level		First GPU-specialized parameter server, reduce latency	2016	[130]	✓		
		Layer swapping and recomputing; out-of-core training behavior analyzing, reduce GPU memory usage	2020	[131]	✓		
		Eliminating memory redundancies with Zero Redundancy Optimizer, reduce memory usage	2020	[260]			✓
		Shared memory, minimum vertex-cut graph partitioning algorithm, delayedupdate algorithms. Reduce latency	2021	[261]	✓		
		Sharing global memory among concurrent jobs, reduce training time	2021	[132]			✓
		A symbolic profiler to estimate memory and computing statistics to improve distributed training efficiency	2023	[251]			✓

TABLE VII: Summary of related inter-device controllers for distributed DL training.

Category		Focus level	Context awareness	Controller	Optimizated performance	Year	Ref.
Inter-device controller level	Federated learning (with privacy concern)	Resource-friendly algorithm level including data distribution and device training latency	Heterogeneous device resources, sample importance	Greedy heuristic	Training time	2021	[265]
		Resource-friendly algorithm level including device conditions	Network bandwidth, heterogeneous devices	Synchronization scheduler	Energy consumption, training time, accuracy	2022	[266]
	Distributed learning (without privacy concern)	Resource-friendly algorithm level including device conditions	Network throughput, computing resources	/	Energy consumption, training time	2022	[133]
		Resource-friendly algorithm level including memory estimation and non-contiguous allocations of DL layers	Memory budget, network bandwidth	Dynamic programming	Execution time	2022	[256]
		Model-adaptive system scheduling including checkpoints in computation graph	Memory budget	Markov Chain Monte Carlo search algorithm	Execution time	2020	[267]
		Model-adaptive system scheduling including checkpoints in computation graph	Memory budget	Dynamic programming	The number of recomputation	2020	[268]
		Model-adaptive system scheduling including model statistics collection and computation graph static planning	Memory budget	/	Execution time	2023	[251]

based compiler that implements memory-aware automated parallelization and provides near-real-time memory and computing statistics. MAP can build distributed execution plans with high training efficiency. Because it involves a symbolic profiler to swiftly collect memory and computation overhead, a cluster detector to gather cluster hardware performance and topology, and a tense layout manager.

3) Inter-device cross-level controller: Like on-device DL training, distributed DL training also benefits from the context-aware controller. On the one hand, imbalances in completion time among different devices result in significant synchronization overheads. On the other hand, DL training parallelism depends on the degree of the decoupled model layer, channel, or operator. Table VII summarizes some prior explorations. In federated learning, the disparate computational capabilities of the devices involved can result in differences in training time, which can impact the overall system efficiency. Lai *et al.* [265] introduced Oort, a method that enhances the performance of federated training and testing through guided participant selection. Oort employs a synchronous federated learning scheme, selecting participants with similar device performance for training in each round to mitigate the impact of hardware differences among devices. To further reduce the impact of heterogeneous devices, Sun *et al.* [266] proposed FedSEA, a semi-asynchronous FL framework for extremely heterogeneous devices. FedSEA adjusts training parameters for device heterogeneity and balances the trade-off between waiting costs and data benefits. MemFlow [267] jointly optimizes memory usage and computation time when searching for an optimal parallelism training strategy. In detail, it adopts a Markov chain monte Carlo search algorithm to select the most suitable degrees of recomputation. Olivier *et al.* [268] employed memory-aware scheduling and automatic differentiation to execute a back-propagation graph within the bounded memory requirement at the cost of computation.

Recently, Liu *et al.* [251] proposed MAP to search for the optimal distributed training strategy in two ways, *i.e.*, intra-operator parallelism and activation checkpoint. Also, to automate intra-operator parallel training for large models, MAP adopted an efficient strategy using a two-stage solver and recompiled it into a module instance. However, the cross-level adaptive controller in a broader space for distributed DL training is still underexplored. According to the dynamically available resources of different AIoT devices in the distributed system without privacy concerns, the split data and model sizes can be dynamically adjusted for diverse devices. To better

TABLE VIII: Comparison of two types of resource-efficient DL engines for AIoT.

Type	Engine	DL model parser	DL model interpreter	DL model optimizer	DL model compiler
Compiled engine	TVM [9]	✓	✓	✓	✓
	OneDNN [35]		✓	✓	✓
	TensorflowXLA [11]	✓		✓	✓
	TensorflowRuntime [10]	✓		✓	✓
Interpreted engine	TensorflowLite [26]	✓	✓	✓	
	CMix-NN [34]		✓	✓	
	CMSIS-NN [269]	✓	✓	✓	

control different devices' training cycles and synchronization strategies, Samikwa *et al.* [133] introduced resource-aware split-learning (ARES) for adaptive DL distributed training. ARES presented the device-oriented model partition method, which adaptively deploys tasks for heterogeneous devices to reduce the impact of stragglers. To address insufficient memory issues on individual end devices, Beaumont *et al.* [256] uses a dynamic programming-based heuristic to find the best strategy for distributed memory allocation.

Discussion. The cross-layer optimization of DL training is essential for achieving efficient and effective AIoT deployments. While both DL training and DL inference (§ III) can benefit from similar optimization approaches and automated controllers, training poses additional constraints and challenges due to higher computational and storage requirements and increased sensitivity to errors. On one hand, both DL training and DL inference share the same cross-layer optimization space and require automated controllers, whether implemented on-device or in a distributed manner. Table IV presents a range of individual optimization techniques that can be employed for DL training task optimization in AIoT systems. Similarly, the performance of optimization techniques can vary significantly, necessitating careful selection and exploration of their combined usage on AIoT devices. On the other hand, DL training has higher computational and storage requirements compared to DL inference, along with increased sensitivity to errors due to multiple iterations. This implies that DL training imposes more constraints and challenges compared to DL inference.

V. Resource-efficient AIoT Applications

This section briefly introduces some enabling systems and the potential application scenarios in AIoT.

A. DL Engines for AIoT

The engine for resource-efficient DL deployment at AIoT devices mainly aims at improving the flexibility of memory scheduling, optimizing the computational efficiency of operators, and enhancing the underlying adaptability to heterogeneous devices. It creates a paradigm shift in how operators are carried out. As shown in Figure 28, they include *interpreted* and *compiled engines*,We demonstrated how these two different deep learning engines play a role in deploying models to devices.

Interpreted engines generally include the DL model parser, interpreter, and optimizer. The DL model parser is responsible for reading and parsing the model file and converting it into a format suitable for processing by the interpreter. The DL model optimizer is responsible for transforming the original DL model into an equivalent model with more efficient inference/training speed; The DL model interpreter accepts the input data from the application scenarios and executes the corresponding internal operators of the DNN in sequence according to the model architecture and target device's hardware schedule configurations to finally output results.

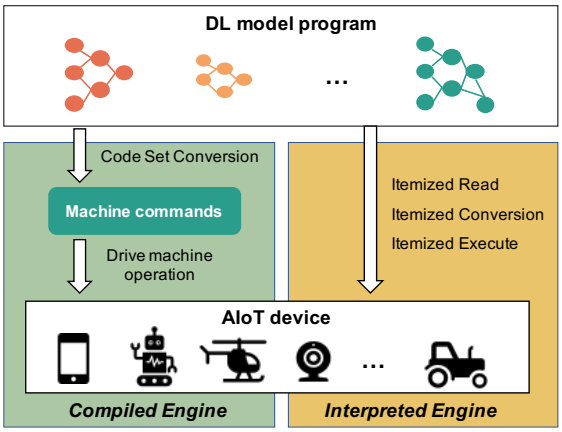


Fig. 28: Difference of interpreted and compiled engines.

Compiled engines mainly involve the DL model parser and compiler. The role of the model parser is the same as that of interpreted engines. The compiler transforms models into machine code that can be directly processed by the target deployment platform (such as CPU, GPU, *etc.* Also, it can apply various optimization methods during the compilation process to improve the operating efficiency of machine codes, such as automatic computing graph scheduling [9].

Table VIII summarizes the state-of-the-art DL deployment engines on AIoT devices. Recent representative compiled inference engines mainly include TVM [9], oneDNN [35], TensorFlow XLA [11], TensorFlow Runtime [10], *et al.*. Different engines focus on different aspects. For example, TVM [9] is a cross-platform DL development framework. Compared with commonly used frameworks such as Tensorflow and Pytorch, TVM optimizes DNN operators at the computation graph level by converting operators into descriptions of tensor changes, generating code, and transmitting it to the CUDA compiler. TVM does not rely on a specific framework's compute library and can be deployed on diverse hardware platforms. Meanwhile, TVM supports integrating new operators. Unlike the cross-platform characteristics of TVM, OneDNN [35] emphasizes on DL performance optimization

with Intel-architecture processors, Intel-architecture graphics cards, and Xe-architecture graphics cards. The oneDNN engine executes DNN primitives to process data in several memory objects, reducing memory usage of DL inference. The primitives are objects encapsulating specific computations, *e.g.*, forward convolution, data transformation, *etc.* Compared to purely functional operations (*e.g.*, conv), primitives can be specialized for a subset of input parameters. Both TVM and OneDNN are compatible with various DL frameworks.

Some other engines are deeply optimized for one framework, such as Tensorflow XLA [11] and Tensorflow Runtime [10]. TensorflowXLA [11] is a linear algebra compiler engine for the TensorFlow framework. In the regular TensorFlow framework, when a program runs, it invokes a pre-compiled GPU kernel for each operation, causing the computing kernel redundancy problem. TensorflowXLA solve this problem by compiling the computation graph of each DNN into a series of specially generated computation kernels to speed up DL inference and improve memory usage with model-specific information. Tensorflow Runtime (TFRT) [10] mainly optimizes the DL inference for specific hardware in various fields to enable scalability. It implements efficient execution of the computing kernel through specific primitives on the underlying devices. Meanwhile, it optimizes the parallel operation of the existing graph and reduces the synchronization overhead. Also, TFRT provides a lightweight just-in-time operator distribution stack for asynchronous API calls to improve computing efficiency.

TFlite [26] is a lightweight engine that is mainly applied to DL inference on mobile devices. TFlite not only provides a series of core operators according to the requirements of mobile platforms but also supports custom operators. And TFlite defines a new DNN file format, removing the parsing step before revisiting data, and greatly reduces the memory footprint of the code. In addition, TFlite designed an optimized interpreter that uses static graphic sorting and a custom (less dynamic) memory allocator to ensure minimal load, and execution latency. CMix-NN [34] is a framework specifically supporting DNN inference on MCU. Compared with other frameworks, CMix-NN focuses on effectively compressing DNNs. Specifically, CMix-NN supports DNN quantization strategies for diverse DNN layer, filter channel, and activation. CMSIS-NN [269] is an edge DNN inference framework for the internet of things (IOT) scenarios. It can directly interact with the underlying hardware, improving computing efficiency by up to $5\times$ in the MCU tests.

Discussion. Building upon the aforementioned techniques, engine systems can assist the algorithm layer in achieving cross-layer optimization of AIoT systems. Compiled engines offer advantages such as reduced memory consumption and simplified deployment, while interpreted engines excel in adapting to various hardware architectures at runtime. The choice between them depends on the specific requirements and constraints of the AIoT system. Compiled engines, with their lower memory footprint and absence of additional graph interpretation, offer improved support for heterogeneous AIoT devices. However, their runtime and dynamic adaptation capabilities remain limited. On the other hand, interpreted engines excel in executing DL inference and training tasks across diverse AIoT devices. They possess the ability to interpret the computation graph based on specific hardware instructions during runtime. This flexibility allows for efficient utilization of the underlying hardware resources, enabling optimal performance across different AIoT device configurations.

B. Resource-efficient AIoT system for Diverse Applications

The remarkable success of DL has fostered a growing number of intelligent applications/services on AIoT devices [248], [282]–[285]. Table IX summarizes three typical AIoT applications, *e.g.*, image classification, semantic segmentation, and speech recognition. And we note that the algorithm-system co-design that jointly optimizes the resource-friendly DL models and model-adaptive resource scheduling can improve the runtime resource availability and thus pushes the limit of performance-resource tradeoff set by standalone levels.

1) Cross-level optimization for image classification: Image classification has a wide range of applications, including object classification [286], human face recognition [287], remote-sensing image recognition [288], image spectrum analysis [289] *etc.* There are many attempts at the algorithm and system scheduling levels. Qian *et al.* [290] proposed the recurrent aggregation operator (ReX) to extract informative information and avoid memory-intensive large-scale early activations. ReX integrates important features of intermediate activations by using two RNNs and compresses them into a low-dimensional vector, which greatly reduces the memory footprint in the early exit module.

2) Cross-level optimization for semantic segmentation: In contrast to image classification, which only requires a single label for the entire image, semantic segmentation requires labels for each pixel. It is the pixel-level segmentation of different objects. Therefore, semantic segmentation is more memory exhaustive than image classification, especially on large images [291]. To realize on-device semantic segmentation on the AIoT device, Jin *et al.* [277] designed an adaptive built-in memory module to decrease memory usage. We note that cross-level algorithm and system scheduling co-design will be beneficial. Wang *et al.* [275] presents Lednet based on an asymmetric encoding and decoding structure. And they designed two new operators, *i.e.*, channel splitting and shuffling, which reduced the computing cost of the entire network and improved the computing speed.

3) Cross-level optimization for speech recognition: Speech recognition has a broad application area, such as smart home [292], [293] and intelligent driving [294], [295]. Diverse studies have investigated compressing memory for various aspects of speech recognition applications, including data separation [296], lightweight models [280], [281], and system deployment [297]. For example, Shangguan *et al.* [278] supported the lightweight conversion of various speech recognition models, such as RNN and LSTM, by integrating different computational graph optimization techniques. Han *et al.* [298] proposed load-balance-aware pruning to ensure high hardware utilization. And they design a system scheduler that encodes the compressed model to multiple PEs for parallelism and schedules the complicated LSTM data flow.

TABLE IX: Summary of resource-efficient AIoT system optimization for enabling diverse resource-aware applications.

Applications	Technique highlight for improving resource efficiency	Focus level	Optimization	Year	Ref.
Image classification	Hyperspectral analysis, less parameters	Computation graph	Improve accuracy	2020	[270]
	Quickly reduce the size of the image	DL model	Reduce parameters	2020	[271]
	Reduce early activation	Operetor	Increase inputs	2021	[272]
	Quantum computing,few memory	Inter-device controller	Reduce data storage	2022	[273]
Semantic segmentation	Space pyramid	DL model	Improve computing speed	2019	[274]
	Asymmetric codec structure	Computation graph	Reduce memory	2019	[275]
	Trapezoidal up-sampling	Operator	Improve computing speed	2020	[276]
	Built-in memory module	Memory scheduling	Improve accuracy	2021	[277]
Speech recognition	End-to-end neural network architecture	Computation graph	Reduce parameters	2019	[278]
	Finite-size beam search decoding	Computation graph	Improve accuracy	2020	[279]
	Low rank matrix substitution	DL model	Reduce parameters	2020	[280]
	Streaming oriented speech separation technology	Inter-device controller	Improve computing speed	2020	[281]

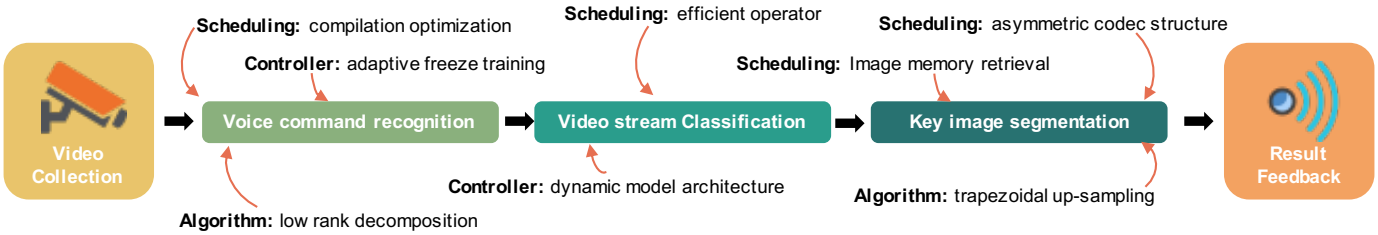


Fig. 29: An example of cross-level optimization for resource-efficient video analysis in smart city scenarios.

4) AIoT-powered application scenarios: Above advances in applications have driven increasing solutions in various AIoT scenarios, including smart homes [299]–[302], smart factories [303]–[305], and smart cities [306]–[308].

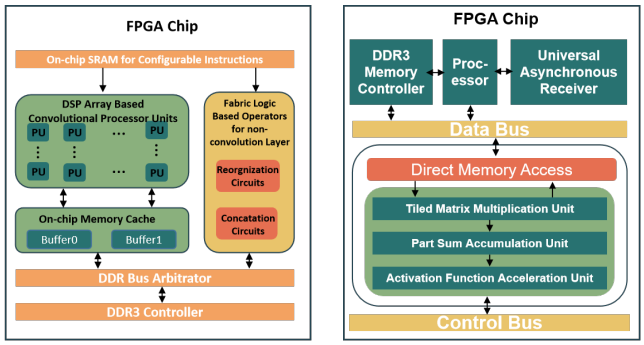
Smart home. Current smart home scenarios involve video/voice recognition and environmental awareness to realize automated appliance operations, *e.g.*, curtain controlling and TV turning on/off. Deeperthings [90] proposed a collaborative optimization framework at three levels: communication, computing, and memory usage. By integrating communication and perception layers to achieve cross-layer overall optimization and balancing memory usage between different devices, the efficient inference is ultimately achieved in resource-constrained situations. Given the diversity and complexity of scenarios, the resource-efficient AIoT system deployed in open environments should dynamically adjust itself. Also, cross-level optimization must be jointly explored.

Smart city. In urban construction and city life, resource-efficient AIoT systems have stimulated plenty of applications, *e.g.*, traffic flow prediction [303], street map estimation [304], and air quality prediction [305]. Specifically, accurate traffic flow prediction serves lane planning [309] and indicator time regulation [310]. As shown in Figure 29, We used monitoring as an example to analyze how optimization technologies at different levels are applied to the systems of smart cities Summarily, resource-efficient AIoT systems in smart cities usually comprise heterogeneous devices with a large physical span. To ensure efficient collaboration, it is necessary to provide unified management of cross-device resources at the algorithm and underlying system levels.

Smart industry. Resource-efficient AIoT systems are gradually integrated into engineering management and process optimization to in the industrial field. The specific functionality includes product defect detection, manufacturing process optimization, predictive operation and maintenance, equipment failure warning, production process planning, *etc.* To avoid the fragmented ecosystem of DL models in AIoT industry, Ren *et al.* [311] proposed a framework using Semantic Web technologies to enable the joint management of TinyML models and IoT devices at scale, from modeling information to discovering possible combinations and benchmarking, and eventually facilitate TinyML component exchange and reuse.

C. Resource-efficient AIoT System on Heterogeneous Devices

Diverse AIoT devices are always heterogeneous regarding memory, computing, and battery resources. And it is non-trivial to deploy resource-efficient AIoT system on heterogeneous devices. Specifically, we select three types of representative AIoT devices, *i.e.*, the cheapest but with extremely constrained MCU devices, the most ubiquitous ARM-based mobile devices (*e.g.*, smartphones, wearables, robots), and typical GPU-based edge servers with weak resources (*e.g.*, NVIDIA DGX, HPE ProLiant DL380 Gen10). As an indispensable interface, smartphones often serve as the control center of AIoT clusters. The MCU is very cheap but has extremely limited computing and memory resources. And the FPGA chip stands out with the advantages of high flexibility, low power consumption, and strong expansibility. Table X summarizes how to optimize resource-efficient AIoT systems in MCU, ARM, and FPGA devices.



(a) DLAU (b) WPU

Fig. 30: Two different FPGA architectures for convolution acceleration, in which DLAU mainly adopts a three-layer pipeline, and WPU uses a sparse network design.

1) Resource-efficient AIoT system on MCUs: MCU is widely used for simple tasks in autopilot, medical care, office equipment, *etc.* And its small size and low cost make it suitable for large-scale deployment, *e.g.*, the daily inspection of the lake surface [312]. However, the MCU's tightly-limited memory resources constrain the efficient execution of DL inference/training tasks. Many studies [12], [71], [123], [226], [313]–[317] have been conducted for DL deployment on MCUs. SpArSe [318] is a network architecture search (NAS) framework designed for DL inference on MCUs. Micronets [313] also leverage NAS for specifying DNN architectures. To map the MCU-imposed memory, latency, and energy constraints to the NAS search framework, they found an important basis, *i.e.*, model latency varies linearly with the model operation (op) counts before searching models in space. And they used these discoveries in NAS to reduce memory utilization and operands. μNAS [71] is another NAS system for MCNS, emphasizing RAM size, memory size, and processor speed. MCUNets [12] integrates Tiny NAS and Tiny Engine to optimize the resource-efficient AIoT systems on MCUs. They find that for a DNN with the same size, the larger the amount of computation, the higher the accuracy. With this insight, Tiny NAS greatly reduces the search space and improves search efficiency. TinyEngine is a compilation engine that reduces the memory occupation in DNN operation by optimizing the loop operation. Facing the huge memory cost of the first few CNN layers, MCUNetsV2 [123] makes longitudinal cutting to execute a small part of models and finally integrate the results of all cut parts.

2) Resource-efficient AIoT system on ARMs: Compared to MCUs, ARM-based devices have more powerful computing power and are widely used in mobile devices such as robots and drones. To accelerate DNN inference/training on ARM devices, many researches [240], [320], [321], [323], [327]–[330] have been proposed. It is reported that the conv layers in DNNs consume most of the computation due to the high computational amount [331]. Huang *et al.* [319] realized the parallel conv computing based on the fast Fourier transform, which optimized the memory occupation and reduced the latency of multi-core parallel. To further improve non-uniform memory access in many-core CPUs, Huang *et al.* [322] proposed a fast Fourier convolution method based on NUMA awareness, conducting data rearrangement and parallelizing complex matrix multiplications to reduce remote memory accesses and thus improves the calculation of CNNs. Meng *et al.* [320] proposed FastConv, a template-based open-source library for automatic code generation, which can automatically generate high-performance CNN kernel and improve the performance of the conv layers on ARM devices. Meanwhile, Li *et al.* [321] focued on maximizing the parallelism of the algorithm to fully utilize the multiple processing cores available on modern ARM CPUs. They propose several optimizations, including a parallel tile processing scheme, a memory layout optimization, and an efficient data rearrangement technique. Zhou *et al.* [323] optimized convolution layers through pipeline strategy. It computed the 3×3 convolution on the ARM CPU by means of a single instruction and multiple data. And it improved the computational efficiency by increasing the data reuse rate. Tencent *et al.* [240] presented NCNN framework, an ARM-based DNN optimization framework that integrates various memory management techniques. It utilizes multi-core parallel to accelerate the DNN calculation. The cross-platform characteristics of NCNN greatly benefit users in transplanting the DL models to the AIoT terminals and reducing the latency and memory occupation of DNNs.

3) Resource-efficient AIoT system on FGPAs: FGPA is developed based on programmable array logic and universal array logic. The main feature of FGPAs is that they are customizable and can independently change the circuit structure and expand the chips according to the amount and way of calculation [203], [324]. When deploying DNNs on FPGAs, such special features can maximize computational efficiency and reduce energy costs. Therefore, in AIoT application scenarios, FPGA is becoming a promising intelligent perception, computing, and control platform [332]. Existing efforts [325], [326], [333]–[336] have begun to study resource-efficient DNN deployments with FGPAs. Wang *et al.* [324] proposed DLAU, a scalable accelerator architecture for the large-scale deployment of DNNs on FGPA. In previous studies, the DL acceleration mainly includes loop unrolling [337], tiling [189], switching [338], *etc.* These methods, however, without hardware redesigning, can hardly give full play to the programmable characteristics of FGPA. To this end, Ma *et al.* [203] designed the specific data stream to accelerate DL execution, as shown in Figure 30(a). To optimize memory accesses, they minimize data traffic and improve computation performance through quantitative analysis of multiple design variables, *e.g.*, energy, and latency. Xie *et al.* [325] proposed the DL model construction algorithm and the accelerator for realizing sparse data selection logic on FGPAs, as shown in Figure 30(b). Most of the existing work requires the off-chip DDR memory to store parameters and the expensive DSP module for DL computation on FGPAs. To overcome this issue, Meng *et al.* [326] proposed FixyFGPA, a DL inference accelerator, which supports high sparsity and low precision computation by integrating dense and sparse computation units in FPGAs. Still, it performs poorly in complex computation. Peng *et al.* [333] presents a novel two-position accelerator by

TABLE X: Summary of resource-efficient system software over heterogeneous AIoT devices.

Device type	Technique highlight	Focus level	Resource efficiency improvements	Year	Ref.
MCU	Sparse architecture search, NN-structure pruning	Computation graph	Reduce memory	2019	[226]
	Constraint mapping, NAS	DL model	Reduce memory, reduce latency, reduce energy	2021	[313]
	Matrix multiplication optimization	DL model	Reduce memory,reduce latency	2021	[71]
	Cooperation of NAS and Engine, calculation library optimization	Memory scheduling	Reduce Memory	2020	[12]
	Patch-to-patch inference, optimization of receptive field	Operator	Reduce memory	2021	[123]
ARM	Parallel convolution algorithm based on Fast Fourier transform	Compiler	Reduce memory, reduce latency	2017	[240]
	Data rearrangement, complex matrix multiplication	DL model	Reduce memory, reduce latency	2020	[319]
	Code automatically generates open source library	Compiler	Reduce memory, improve accuracy	2022	[320]
	Adjust data layout, convolution based on Winograd	Intra-device controller	Reduce memory, reduce latency	2021	[321]
	Pipeline strategy, improve data reuse	Computation graph	Reduce memory, reduce latency	2021	[322]
	Fine memory management and data structure design	Memory scheduling	Reduce memory, improve accuracy	2022	[323]
FPGA	Pipeline strategy, scalable accelerator architecture	Computation graph	Reduce latency; reduce energy	2016	[324]
	Simpler sparse data selection logic	Memory scheduling	Improve resource utilization	2018	[203]
	Support high sparsity, low precision calculation	Computation graph	Reduce latency	2021	[325]
	Two-bit convolution accelerator	DL model	Reduce latency	2021	[326]

combining the deep complex network with the binary neural network to speed up inference on FGPAs.

VI. Open Issues and Future Directions

This section discusses existing challenges and potential misleading directions related to resource-efficient AIoT systems and enabling technologies.

A. Cross-level AIoT System

Despite various optimization techniques at a single level, *e.g.*, algorithm, computation graph, compiler, operator, hardware instruction, without considering their cooperation, we discuss the open issues as below.

(i) Cross-level co-design. Prior efforts in cross-level co-design mainly include algorithm-hardware [339]–[341], compiler-hardware [342], [343], and algorithm-system [12], [344], [345] co-design. The algorithm-system co-design is a promising way to address the increasing complexity of DL models for complicated application problems and optimize the runtime execution of those models on AIoT hardware. While the hardware re-design in the first two types is costly for existing AIoT applications and suitable for brand-new construction. Some of the key open issues in cross-level algorithm-system co-design include: First, *Runtime resource-efficient algorithm*. The challenge is how to propagate the feedback of the runtime system execution to the algorithm design. State-of-the-art on algorithm-system co-design like PCONV [346], PatDNN [344], and CoCoPIE [345] simultaneously optimize the model compression algorithms and runtime operator/memory scheduling mechanisms. They leverage some fixed resource-friendly patterns to guide model design. Specifically, DL models are specified with a specific shape to maximize and sustain instruction-level and thread-level parallelism. Second, *Model-adaptive runtime compiler/engine*. Most of the existing works at the compiler and engine level for operator/memory scheduling only manually optimize partial factors. For example, the co-design MCUNet is composed of TinyNAS and TinyEngine. TinyNAS searches for the most efficient model architecture running on TinyEngine. At the same time, the TinyEngine library generates codes for the network search space of TinyNAS to eliminate instruction and memory redundancy. Moreover, it is non-trivial to make the compiler/engine to be compatible with various hardware backends of AIoT devices. They consist of multiple processing units with different architectures and capabilities, *e.g.*, Jetsons [347], Raspberry Pis [144], FPGAs [348].

(ii) Cross-level adaptive controller. Given heterogeneous AIoT devices and diverse performance demands, it is necessary to automate the adaptive optimization control across multiple levels, *e.g.*, DL model, operator, memory allocation, compiler, engine, and hardware instructions. The adaptive controller at the algorithm level with automatic search, *e.g.*, neural architecture search (NAS), is widely explored [349]–[351]. While the adaptive controller across algorithm and system levels is less explored. The challenges are from three aspects: First, the *combined search space* should consider the complex cross-level dependency and collaborations in different techniques, as mentioned in III and IV. Second, the *performance validation* of candidates is non-trivial because the cross-level algorithm and system co-design cover multiple phases, *i.e.*, offline model design and parameter training, runtime compiler, and accuracy testing. Integrating their validation and verification together is desired yet challenging, especially as the design becomes more complex. Third, the *runtime search algorithm*, *i.e.*, boosting the efficiency and efficacy of solving constrained multi-objective optimization problems, is a long-term open problem [352] [353].

B. Context-aware AIoT System Evolution

As discussed in § II-E, to ensure that the deployed AIoT system can maintain continuous and stable high-quality services in the long-term life cycle, the resource demands of the AIoT system should be evolvable. That is, the AIoT system can be compatible with the *dynamic nature of the deployment context* (*e.g.*, the dynamic resource availability, diverse performance

demands) and integrates the adaptation loop to adjust all the system blocks. The AIoT system evolution requirements include *DL model structure scaling* and *weight retraining*. The former is always caused by mismatching between model resource demand and device resource supply, and the latter is usually required by model accuracy degradation in live sensing data. In particular, the evolvable AIoT system may need to deal with the following problems.

(i) Resource availability monitoring. The key resources affecting DL model deployment (*e.g.*, memory budget) and performance outcome (*e.g.*, latency, energy efficiency) in AIoT devices include memory, computing, and battery. All of them exhibit high dynamics over time [8]. For example, the battery-powered device will gradually consume energy as the device runs, and the available memory or computing resources will be encroached upon by other applications/tasks running on a device. The dynamic nature of available resources in AIoT devices is also related to the operating system (OS) resource scheduling. A fast and accurate resource monitor that can interact with heterogeneous and cross-platform OS is needed.

(ii) Resource demand prediction and performance profiling of DL inference/training tasks. The resource demand of deployed AIoT systems contains energy cost, computation, and memory usage. And systematic formulation of these resource demands can predict the most suitable DL inference/training configurations in advance. The AIoT system performance involves DL inference accuracy, inference latency, and training convergence latency. The inference accuracy is coupled with the DNN parameter weights, and dramatic accuracy drop trigger DNN retraining for weight evolving. As mentioned in § II-E, the total computation amount and memory usage for DL inference and training tasks can be directly calculated by the DNN structure and training configurations (*e.g.*, iteration number, batch size). While the prediction of energy demand and latency is non-trivial since its measurement is not straightforward. They heavily depend on the underlying operator scheduling, data flow, and memory bandwidth bound [146]. Measuring energy cost and latency in real-world devices is infeasible or too costly [148]. However, it is highly desirable for many tasks, *e.g.*, searching for the most suitable DNN structure with latency demands from a vast space.

(iii) Dynamic context awareness and system evolution trigger. The dynamic context awareness block detects the mismatch between resource supply and demand or the dramatic accuracy drop to trigger the adaptation block. The triggering station for resource supply-demand mismatch can be further modeled as the noticeable context changes. We note that the onset of the accuracy drop is not always the optimal trigger time point for DNN retraining, which may increase unnecessary retraining workload and even lag some necessary evolution tasks. A suitable trigger is also desired for evolving efficiency and efficacy.

(iv) Automated system loop. The self-evolutionary AIoT system needs an automated loop consisting of the *resource monitor*, the *runtime resource demand and performance profiler*, the *evolution trigger*, and the *optimizer*. The resource monitor tracks the memory/computing resource supply of the available AIoT devices. The resource demand profiler predicts the DL inference tasks' memory, computing, and energy resource demands with the current configurations. If the resource demand exceeds the supply or the accuracy drop exceeds a pre-defined threshold, the evolution trigger notifies the optimizer, adjusting the DL inference/training configurations. The automated control loop routinely checks for system changes and performs on-demand evolution.

C. Distributed AIoT Resource Aggregation in DL inference/training Tasks

Efficiently aggregating memory and computing resources within the networked AIoT system for seamless communication and the provision of complex services is an active and research-intensive domain, which has several open issues.

- How to uniformly manage heterogeneous resources (*e.g.*, SRAM, DRAM, CPU/GPU, battery) within the networked AIoT system? They should be organized and managed in a unified manner to enable efficient DNN training or inference tasks.
- How to combine operator scheduling, memory allocation, and hardware instruction mechanisms within the networked AIoT system. Specifically, the computing power of AIoT devices is subject to the *barrel effect* of memory, computing, and battery resources. The shortest board of the barrel restricts the overall computing capability. Thus, monitoring the AIoT device resources with *normalized assessment necessary* is necessary.
- How to balance multiple performance goals in the resource aggregation process (*e.g.*, energy efficiency and latency)? We need to establish the hierarchical dependence between several internal factors (*e.g.*, DL model, operator, memory, instructions) of the AIoT system and the external environment (*e.g.*, input).
- How to develop hybrid distributed resource aggregation methods that combine both synchronous and asynchronous communication schemes to balance the trade-off between convergence speed and system stability.
- How to deal with the opportunistic connection problem of AIoT devices for guaranteeing resource availability?

Advanced research in areas such as edge intelligence [354] and federated learning [106] intersects with some of the issues mentioned above, but it cannot comprehensively address them. To tackle these challenges, it is important to decouple resource aggregation methods from the DL inference/training tasks and develop them as a general-purpose middleware function that can actively perceive, analyze, and select AIoT resources to meet diverse demands.

D. Intelligence Enhancement in Distributed DL Training

AIoT devices have been extensively deployed in numerous domains. Their wide array of sensors allows for the collection of massive amounts of data, facilitating the execution of DL tasks and enabling different intelligent applications. In practical AIoT systems, a large number of distributed devices continuously sense the environment and hold accumulated datasets. However, real-world scenarios present several challenges that must be addressed to ensure reliable and efficient

data transmission, synchronization, and coordination among these devices. These challenges include:

(i) Intrinsic linkages of distributed holding datasets.. To fully leverage the distributed devices' computing power and accumulated datasets, studying the intrinsic linkages between distributed datasets in different AIoT sub-clusters and designing corresponding distributed DL training systems is essential. In particular, the inherent correlation, redundancy, and hysteresis of distributed datasets can drive the design of distributed training systems, leading to high-quality, responsive, and low-cost distributed DL training.

(ii) Temporal collaboration of distributed devices. In real-world AIoT systems, each device may have different training power and speed. Asynchronous communication methods have been proposed to mitigate this problem to some extent [355], but extreme temporal differences still pose a significant challenge to participant collaboration. Specifically, the challenge is effectively promoting collaboration among participant devices with different temporal patterns. For example, existing distributed DL training systems often partition the dataset by data category to highlight the Non-IID nature of the data [356]. However, the data distribution collected by AIoT devices is clearly more diverse and challenging. Possible directions for distributed DL training can consider:

- The temporal correlation of the distributed data at diverse sub-clusters of AIoT devices varies. Distributed DL model/data aggregation and communication mechanisms in AIoT system's distributed learning for syn-/asynchronic datasets/models should be different.
- The spatial correlation of the distributed data in distributed DNN training, *e.g.*, physically nearby devices have the potential for better collaboration gains.
- We can divide the participating AIoT devices into diverse roles in distributed DL training, *e.g.*, collaborators, competitors, and supervisors.

E. Inference and Training Task Balance in Resource-constrained AIoT Devices

Resouce-constrained AIoT devices always load lightweight Dl models, such as compressed DL models. However, DL models with shallow or sparse structures are highly susceptible to data drift, which occurs when the live data stream captured by the devices diverges from the data used for training, leading to a drop in accuracy in real-world applications. Some efforts [357]–[359] have proven that DL models deployed in AIoT devices should be continuously retrained using newly captured live data to maintain accuracy.

However, AIoT devices provide limited resources to execute computation. Introducing training tasks into AIoT devices will likely *deprive resources of the inference tasks for training*, leading to decreased inference performance. The more resources we allocate to the task of DL retraining, the faster the training process will be, and we can obtain a high-accuracy inference model earlier. However, the limited resources allocated to the inference task during the training process may decrease the overall performance of DL inference. On the other hand, allocating too few resources to the retraining task may slow down the retraining speed and delay the accuracy improvement of the updated model, which could impact the timely inference accuracy gain. Therefore, it is crucial to trade off the accuracy improvement from training with real-time inference performance. We discuss open issues in more detail:

- How to allocate resources for maximizing the inference accuracy when the training task deprives a certain amount of memory, regarding the tunable data flow in memory units. For example, we can share memory resources between the inference and training tasks via memory reallocation, recomputation, and swapping techniques.
- How to select promising DL training tasks and allocate computing resources among them in AIoT devices.
- How to select DL training configurations (*e.g.*, epoch) and inference configurations (*e.g.*, sampling rate) to maximize inference accuracy with minimum resource usage.

Making accurate decisions on resource allocation and performance assessment is challenging in advance DL training. Furthermore, due to the dynamic nature of AIoT context, the optimal resources and configurations may change over time. The best decision at the current training may become sub-optimal for future training phases.

F. Memory-computation Feedback and Joint-optimization

Memory and computation cost of DL models are two crucial metrics tunable in AIoT systems to affect overall performance, *e.g.*, delay, and energy cost. However, previous studies have shown that memory and computation optimization are often conflicting goals. For instance, recomputation techniques can save memory space by discarding and recalculating intermediate activations, but they can also introduce extra computing delay [120], [121], [223]. Further research is needed to balance these two metrics best and enable memory-computation joint optimization in AIoT systems.

(i) Near- and in-memory computing. The underlying memory schedule techniques have not kept up with computation optimization advances in latency and energy reduction over the years, referred to as the memory wall [360]. The development of DL has exacerbated this problem as the frequent movement of a large amount of data, including input data and intermediate activations between memory and compute units. They seriously impacted the latency and energy cost of DL training and inference tasks. To address this issue, some studies [361]–[365] physically relocate compute units (multi-core, GPU, FPGA) closer to memory to reduce data transport costs. Near-memory and in-memory computing embed computation in the memory array. As compute units become more intimately connected with memory, finer-grained parallelism can be proposed to improve energy efficiency and latency [366]–[369]. However, these compute units placed next to memory have problems such as less supporting computing types and weak computing power. Thus, designing at the system level, rather than the hardware level, to deploy data-intensive operators (such as ReLU) in near-memory compute units is promising, especially for ubiquitous AIoT devices without hardware replacement. DL operators need to access a large amount of data for computing, which causes considerable

data transfer delay and lower computing demands. Also, we can carefully allocate DL operators to the near- and in-memory units to reduce memory access costs and improve computation efficiency at the operator and instructor levels.

(ii) Memory-aware computing in runtime/compiler optimization. Memory-computation joint optimization for AIoT can be achieved by co-designing the DL algorithm and engine. At the algorithm level, we can interleave computation-intensive and data-intensive operators to balance the workload of compute units near memory. At the engine level, we can optimize the computation graph and generate execution code to reduce access delay for data-intensive operators. These can be achieved by optimizing the inner layers of the loop.

VII. Conclusion

Artificial Intelligence of Things (AIoT) combines AI technologies with IoT infrastructures, enhancing the efficiency and efficacy of data analysis. However, due to the heterogeneous and dynamic nature of AIoT hardware, co-designed cross-level AIoT system and adaptive controllers are needed to expand the boundaries of system performance beyond what can be achieved by algorithm-level techniques alone. These co-designed systems can push the boundaries of resource-performance tradeoffs for AIoT. Specifically, the cross-level AIoT system spans on-device and distributed DL training/inference algorithms, computation graphs, operators, memory schedules, hardware instructions, *etc.* With the continuous development of DL technologies and AIoT devices, the AIoT system expands the cyber-physical space to the human space, providing low-cost, high-quality, and inclusive ubiquitous intelligence for a wide range of AIoT application domains. Given the heterogeneous and dynamic nature of AIoT hardware, co-designed cross-level AIoT systems, and adaptive controllers can further expand the boundaries of system performance, including accuracy and resource consumption, beyond what can be achieved with algorithm-level or system-level techniques alone. We hope this survey will raise awareness and stimulate discussion among researchers and developers on AIoT system. This paper elucidates many aspects of distributed device collaboration (*i.e.*, network topology establishment), and data exchange in real-world inference and training tasks. These insights are invaluable for communication researchers, as they provide a deeper understanding of the intricacies involved. More heuristics and insights are needed to ensure resource-efficient AIoT systems. For example, reliable and real-time communication can further enhance network efficiency in AIoT deployments.

Acknowledgments

This work was partially supported by the National Science Fund for Distinguished Young Scholars (62025205) and the National Natural Science Foundation of China (No. 62032020, 62102317).

References

[1] A. Ghosh, D. Chakraborty, and A. Law, "Artificial intelligence in internet of things," *CAAI Transactions on Intelligence Technology*, vol. 3, no. 4, pp. 208–218, 2018.

[2] "Cisco annual internet report (2018–2023) white paper," https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[3] M. U. Hassan, M. H. Rehmani, and J. Chen, "Privacy preservation in blockchain based iot systems: Integration issues, prospects, challenges, and future research directions," *Future Generation Computer Systems*, vol. 97, pp. 512–529, 2019.

[4] 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 arXiv:1704.04861*, 2017.

[5] P. P. Ray, "A review on tinyml: State-of-the-art and prospects," *Journal of King Saud University-Computer and Information Sciences*, 2021.

[6] P. Guo, B. Hu, and W. Hu, "Mistify: Automating dnn model porting for on-device inference at the edge," in *NSDI*, 2021.

[7] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th annual international conference on mobile computing and networking*, 2019, pp. 1–16.

[8] S. Liu, B. Guo, K. Ma, Z. Yu, and J. Du, "Adaspring: Context-adaptive and runtime-evolutionary deep model compression for mobile applications," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 1, pp. 1–22, 2021.

[9] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated {End-to-End} optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[10] tensorflow, "Tensorflowruntime," https://blog.tensorflow.org/2020/04/tfrt-new-tensorflow-runtime.html.

[11] ——, "tensorflowxla," https://www.tensorflow.org/xla?hl=zh-cn.

[12] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han *et al.*, "Mcunet: Tiny deep learning on iot devices," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 711–11 722, 2020.

[13] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "Hardware-aware neural architecture search: Survey and taxonomy." in *IJCAI*, 2021, pp. 4322–4329.

[14] P. Joshi, H. Afli, M. Hasanuzzaman, C. Thapa, and T. Scully, "Enabling deep learning for all-in edge paradigm," *arXiv preprint arXiv:2204.03326*, 2022.

[15] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[16] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[17] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian, "The deep learning compiler: A comprehensive survey," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 708–727, 2020.

[18] K. Zhang, H. Ying, H.-N. Dai, L. Li, Y. Peng, K. Guo, and H. Yu, "Compacting deep neural networks for internet of things: Methods and applications," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 11 935–11 959, 2021.

[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[20] S. Tang, L. Chen, K. He, J. Xia, L. Fan, and A. Nallanathan, "Computational intelligence and deep learning for next-generation edge-enabled industrial iot," *IEEE Transactions on Network Science and Engineering*, 2022.

[21] 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.

[22] P. Nauth, *Embedded intelligent systems*. Walter de Gruyter, 2009.

[23] I. Ghosh, "Aiot: when artificial intelligence meets the internet of things," *Visual Capitalist*, vol. 12, 2020.

[24] "The third wave of ai," https://elleknowsmachines.com/third-wave-of-ai/.

[25] S. S. Saha, S. S. Sandha, and M. Srivastava, "Machine learning for microcontroller-class hardware-a review," *IEEE Sensors Journal*, 2022.

[26] "Tensorflow lite," https://www.tensorflow.org/lite.

[27] "Caffe2," https://caffe2.ai/.

[28] "Pytorch mobile," https://pytorch.org/mobile/home/.

[29] S. Nazir, Y. Ali, N. Ullah, and I. García-Magariño, "Internet of things for healthcare using effects of mobile computing: a systematic literature review," *Wireless Communications and Mobile Computing*, vol. 2019, pp. 1–20, 2019.

[30] G. Cicceri, F. De Vita, D. Bruneo, G. Merlino, and A. Puliafito, "A deep learning approach for pressure ulcer prevention using wearable computing," *Human-centric Computing and Information Sciences*, vol. 10, no. 1, pp. 1–21, 2020.

[31] P. P. Ray, "A review on tinyml: State-of-the-art and prospects," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, 2022.

[32] S. Han. Tinyml project. [Online]. Available: https://tinyml.mit.edu/

[33] J. Mendez, K. Bierzynski, M. Cuéllar, and D. P. Morales, "Edge intelligence: Concepts, architectures, applications and future directions," *ACM Transactions on Embedded Computing Systems*, 2022.

[34] A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 871–875, 2020.

[35] Intel, "onednn," https://github.com/oneapi-src/oneDNN.

[36] T. Wan, B. Shao, S. Ma, Y. Zhou, Q. Li, and Y. Chai, "In-sensor computing: Materials, devices, and integration technologies," *Advanced Materials*, p. 2203830, 2022.

[37] F. Zhou and Y. Chai, "Near-sensor and in-sensor computing," *Nature Electronics*, vol. 3, no. 11, pp. 664–671, 2020.

[38] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256kb memory," *arXiv preprint arXiv:2206.15472*, 2022.

[39] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[40] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, Z. Xu, and T. Kraska, "Superneurons: Dynamic gpu memory management for training deep neural networks," in *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, 2018, pp. 41–53.

[41] T. Li, J. Huang, E. Risinger, and D. Ganesan, "Low-latency speculative inference on distributed multi-modal data streams," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 67–80.

[42] S. Petridis, T. Stafylakis, P. Ma, F. Cai, G. Tzimiropoulos, and M. Pantic, "End-to-end audiovisual speech recognition," in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018, pp. 6548–6552.

[43] Y. Tian, J. Shi, B. Li, Z. Duan, and C. Xu, "Audio-visual event localization in unconstrained videos," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 247–263.

[44] V. Radu, C. Tong, S. Bhattacharya, N. D. Lane, C. Mascolo, M. K. Marina, and F. Kawsar, "Multimodal deep learning for activity and context recognition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, pp. 1–27, 2018.

[45] Z. Ning, P. Dong, X. Wang, M. S. Obaidat, X. Hu, L. Guo, Y. Guo, J. Huang, B. Hu, and Y. Li, "When deep reinforcement learning meets 5g-enabled vehicular networks: A distributed offloading framework for traffic big data," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 2, pp. 1352–1361, 2019.

[46] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *Journal of Network and Computer Applications*, vol. 169, p. 102781, 2020.

[47] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *Acm computing surveys (csur)*, vol. 53, no. 2, pp. 1–33, 2020.

[48] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

[49] J. Tan, C. Wang, B. Li, Q. Li, W. Ouyang, C. Yin, and J. Yan, "Equalization loss for long-tailed object recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 662–11 671.

[50] M. Zhou, Y. Bai, W. Zhang, T. Zhao, and T. Mei, "Look-into-object: Self-supervised structure modeling for object recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 774–11 783.

[51] A. T. Abu-Jassar, Y. M. Al-Sharo, V. Lyashenko, and S. Sotnik, "Some features of classifiers implementation for object recognition in specialized computer systems," *TEM Journal*, vol. 10, no. 4, p. 1645, 2021.

[52] W. Xu, Z. Yang, D. W. K. Ng, M. Levorato, Y. C. Eldar, and M. Debbah, "Edge learning for b5g networks with distributed signal processing: Semantic communication, edge computing, and wireless sensing," *IEEE Journal of Selected Topics in Signal Processing*, 2023.

[53] Y. Yuan, X. Chen, and J. Wang, "Object-contextual representations for semantic segmentation," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*. Springer, 2020, pp. 173–190.

[54] W. Wang, T. Zhou, F. Yu, J. Dai, E. Konukoglu, and L. Van Gool, "Exploring cross-image pixel contrast for semantic segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 7303–7313.

[55] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 077–12 090, 2021.

[56] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. Torr, "Fast online object tracking and segmentation: A unifying approach," in *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, 2019, pp. 1328–1338.

[57] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, "Bytetrack: Multi-object tracking by associating every detection box," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXII*. Springer, 2022, pp. 1–21.

[58] T. Meinhardt, A. Kirillov, L. Leal-Taixe, and C. Feichtenhofer, "Trackformer: Multi-object tracking with transformers," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 8844–8854.

[59] A. Galassi, M. Lippi, and P. Torroni, "Attention in natural language processing," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 10, pp. 4291–4308, 2020.

[60] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[61] S. Wang, L. Hu, Y. Wang, L. Cao, Q. Z. Sheng, and M. Orgun, "Sequential recommender systems: challenges, progress and prospects," *arXiv preprint arXiv:2001.04830*, 2019.

[62] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[63] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–37, 2022.

[64] C. Guo, C. Li, J. Guo, C. C. Loy, J. Hou, S. Kwong, and R. Cong, "Zero-reference deep curve estimation for low-light image enhancement," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1780–1789.

[65] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," *ACM SIGOPS Operating Systems Review*, vol. 33, no. 5, pp. 48–63, 1999.

[66] L. Benini and G. d. Micheli, "System-level power optimization: techniques and tools," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 5, no. 2, pp. 115–192, 2000.

[67] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[68] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[69] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1389–1397.

[70] P. Kaloshin, "Convolutional neural networks compression with low rank and sparse tensor decompositions," *arXiv preprint arXiv:2006.06443*, 2020.

[71] E. Liberis, Ł. Dudziak, and N. D. Lane, "μnas: Constrained neural architecture search for microcontrollers," in *Proceedings of the 1st Workshop on Machine Learning and Systems*, 2021, pp. 70–79.

[72] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 326–335, 2020.

[73] M. Rusci, M. Fariselli, A. Capotondi, and L. Benini, "Leveraging automated mixed-low-precision quantization for tiny edge microcontrollers," in *IoT Streams for Data-Driven Predictive Maintenance and*

IoT, Edge, and Mobile for Embedded Machine Learning. Springer, 2020, pp. 296–308.

[74] S. Liu, J. Du, K. Nan, Z. Zhou, H. Liu, Z. Wang, and Y. Lin, "Adadeep: a usage-driven, automated deep model compression framework for enabling ubiquitous intelligent mobiles," *IEEE Transactions on Mobile Computing*, vol. 20, no. 12, pp. 3282–3297, 2020.

[75] Y. Ding, L. Zhu, Z. Jia, G. Pekhimenko, and S. Han, "Ios: Inter-operator scheduler for cnn acceleration," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 167–180, 2021.

[76] X. Cai, Y. Wang, and L. Zhang, "Optimus: An operator fusion framework for deep neural networks," *ACM Transactions on Embedded Computing Systems (TECS)*, 2022.

[77] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren, "Dnnfusion: accelerating deep neural networks execution with advanced operator fusion," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 883–898.

[78] H. Miao and F. X. Lin, "Enabling large neural networks on tiny microcontrollers with swapping," *arXiv preprint arXiv:2101.08744*, 2021.

[79] J.-C. Huang and T. Leng, "Generalized loop-unrolling: a method for program speedup," in *Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99 (Cat. No. PR00122)*. IEEE, 1999, pp. 244–248.

[80] E.-J. Im and K. Yelick, "Optimizing sparse matrix computations for register reuse in sparsity," in *Computational Science—ICCS 2001: International Conference San Francisco, CA, USA, May 28–30, 2001 Proceedings, Part I 1*. Springer, 2001, pp. 127–136.

[81] M. Boehm, B. Reinwald, D. Hutchison, A. V. Evfimievski, and P. Sen, "On optimizing operator fusion plans for large-scale machine learning in systemml," *arXiv preprint arXiv:1801.00829*, 2018.

[82] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: a system for {Large-Scale} machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[83] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[84] S. Yun, W. Choi, and I.-M. Kim, "Cooperative inference of dnns for delay-and memory-constrained wireless iot systems," *IEEE Internet of Things Journal*, 2022.

[85] W. Wu, P. Yang, W. Zhang, C. Zhou, and X. Shen, "Accuracy-guaranteed collaborative dnn inference in industrial iot via deep reinforcement learning," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4988–4998, 2020.

[86] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint dnn partition deployment and resource allocation for delay-sensitive deep learning inference in iot," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9241–9254, 2020.

[87] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 1396–1401.

[88] J. Mao, Z. Yang, W. Wen, C. Wu, L. Song, K. W. Nixon, X. Chen, H. Li, and Y. Chen, "Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 751–756.

[89] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[90] R. Stahl, A. Hoffman, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Deeperthings: Fully distributed cnn inference on resource-constrained edge devices," *International Journal of Parallel Programming*, vol. 49, no. 4, pp. 600–624, 2021.

[91] W. Zhang, D. Yang, H. Peng, W. Wu, W. Quan, H. Zhang, and X. Shen, "Deep reinforcement learning based resource management for dnn inference in industrial iot," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 7605–7618, 2021.

[92] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[93] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Toward collaborative inferencing of deep neural networks on internet-of-things devices," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4950–4960, 2020.

[94] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "Deepslicing: collaborative and adaptive cnn inference with low latency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2175–2187, 2021.

[95] S. J. Pan, X. Ni, J.-T. Sun, Q. Yang, and Z. Chen, "Cross-domain sentiment classification via spectral feature alignment," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 751–760.

[96] R. Chattopadhyay, Q. Sun, W. Fan, I. Davidson, S. Panchanathan, and J. Ye, "Multisource domain adaptation and its application to early detection of fatigue," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 4, pp. 1–26, 2012.

[97] L. Duan, I. W. Tsang, and D. Xu, "Domain transfer multiple kernel learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 3, pp. 465–479, 2012.

[98] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[99] Q. Wang, O. Fink, L. Van Gool, and D. Dai, "Continual test-time domain adaptation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7201–7211.

[100] K. Shen, R. M. Jones, A. Kumar, S. M. Xie, J. Z. HaoChen, T. Ma, and P. Liang, "Connect, not collapse: Explaining contrastive learning for unsupervised domain adaptation," in *International Conference on Machine Learning*. PMLR, 2022, pp. 19 847–19 878.

[101] R. Gal, O. Patashnik, H. Maron, A. H. Bermano, G. Chechik, and D. Cohen-Or, "Stylegan-nada: Clip-guided domain adaptation of image generators," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pp. 1–13, 2022.

[102] B. Xie, L. Yuan, S. Li, C. H. Liu, X. Cheng, and G. Wang, "Active learning for domain adaptation: An energy-based approach," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8708–8716.

[103] F. Xu, Z. Pan, and R. Xia, "E-commerce product review sentiment classification based on a naïve bayes continuous learning framework," *Information Processing & Management*, vol. 57, no. 5, p. 102221, 2020.

[104] M. Irfan, Z. Jiangbin, M. Iqbal, and M. H. Arif, "A novel lifelong learning model based on cross domain knowledge extraction and transfer to classify underwater images," *Information Sciences*, vol. 552, pp. 80–101, 2021.

[105] J. Yuan, S. E. Liu, A. Shylendra, W. A. Gaviria Rojas, S. Guo, H. Bergeron, S. Li, H.-S. Lee, S. Nasrin, V. K. Sangwan *et al.*, "Reconfigurable mos2 memtransistors for continuous learning in spiking neural networks," *Nano letters*, vol. 21, no. 15, pp. 6432–6440, 2021.

[106] W. Huang, M. Ye, and B. Du, "Learn from others and be yourself in heterogeneous federated learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 143–10 153.

[107] X. Fang and M. Ye, "Robust federated learning with noisy and heterogeneous clients," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 072–10 081.

[108] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[109] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *arXiv preprint arXiv:1711.00436*, 2017.

[110] A. Tjandra, S. Sakti, and S. Nakamura, "Tensor decomposition for compressing recurrent neural network," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[111] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2423–2432.

[112] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce memory, not parameters for efficient on-device learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 285–11 297, 2020.

[113] S. Liu, C. Zheng, Y. Huang, and T. Q. Quek, "Distributed reinforcement learning for privacy-preserving dynamic edge caching," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 3, pp. 749–760, 2022.

[114] J. J. Moon, P. Kapoor, J. H. Lee, M. J. Ham, and H. S. Lee, "Nntrainer: Light-weight on-device training framework," *arXiv preprint arXiv:2206.04688*, 2022.

[115] Z. Deng, C. Xu, Q. Cai, P. Faraboschi, and H. Packard, "Reduced-precision memory value approximation for deep learning," *Hewlett Packard Labs, HPL-2015-100*, 2015.

[116] S. R. Bulo, L. Porzi, and P. Kontschieder, "In-place activated batchnorm for memory-optimized training of dnns," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5639–5647.

[117] W. Jung, D. Jung, B. Kim, S. Lee, W. Rhee, and J. H. Ahn, "Restructuring batch normalization to accelerate cnn training," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 14–26, 2019.

[118] L. Liu, L. Deng, X. Hu, M. Zhu, G. Li, Y. Ding, and Y. Xie, "Dynamic sparse graph for efficient deep learning," *arXiv preprint arXiv:1810.00859*, 2018.

[119] P. Dai, J. Yang, X. Ye, X. Cheng, J. Luo, L. Song, Y. Chen, and W. Zhao, "Sparsetrain: Exploiting dataflow sparsity for efficient convolutional neural networks training," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[120] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016.

[121] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," *Advances in neural information processing systems*, vol. 30, 2017.

[122] A. Jain, A. Phanishayee, J. Mars, L. Tang, and G. Pekhimenko, "Gist: Efficient data encoding for deep neural network training," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 776–789.

[123] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "Memory-efficient patch-based inference for tiny deep learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2346–2358, 2021.

[124] Z. Jia, J. Thomas, T. Warszawski, M. Gao, M. Zaharia, and A. Aiken, "Optimizing dnn computation with relaxed graph substitutions," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 27–39, 2019.

[125] Y. Zhou, S. Roy, A. Abdolrashidi, D. Wong, P. Ma, Q. Xu, H. Liu, P. Phothilimtha, S. Wang, A. Goldie *et al.*, "Transferable graph optimizers for ml compilers," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 844–13 855, 2020.

[126] X. Chen, D. Z. Chen, and X. S. Hu, "modnn: Memory optimal dnn training on gpus," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 13–18.

[127] T. Huang, L. Tao, and J. T. Zhou, "Adaptive precision training for resource constrained devices," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 1403–1408.

[128] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

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

[130] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, "Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server," in *Proceedings of the eleventh european conference on computer systems*, 2016, pp. 1–16.

[131] M. Wahib, H. Zhang, T. T. Nguyen, A. Drozd, J. Domke, L. Zhang, R. Takano, and S. Matsuoka, "Scaling distributed deep learning workloads beyond the memory capacity with karma," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[132] G. Lim, J. Ahn, W. Xiao, Y. Kwon, and M. Jeon, "Zico: Efficient gpu memory sharing for concurrent dnn training." in *USENIX Annual Technical Conference*, 2021, pp. 161–175.

[133] E. Samikwa, A. Di Maio, and T. Braun, "Ares: Adaptive resource-aware split learning for internet of things," *Computer Networks*, vol. 218, p. 109380, 2022.

[134] "Open neural network exchange," https://onnx.ai/.

[135] D. Gudovskiy, A. Hodgkinson, and L. Rigazio, "Dnn feature map compression using learned representation over gf (2)," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.

[136] Y. Wu, Y. Gong, P. Zhao, Y. Li, Z. Zhan, W. Niu, H. Tang, M. Qin, B. Ren, and Y. Wang, "Compiler-aware neural architecture search for on-mobile real-time super-resolution," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XIX*. Springer, 2022, pp. 92–111.

[137] K.-C. Tai, "Constant folding within an expression by semantic attributes," *Computer Languages*, vol. 4, no. 3-4, pp. 131–137, 1979.

[138] S. Glesner and J. O. Blech, "Classifying and formally verifying integer constant folding," *Electronic Notes in Theoretical Computer Science*, vol. 82, no. 2, pp. 410–425, 2004.

[139] J. Cocke, "Global common subexpression elimination," in *Proceedings of a symposium on Compiler optimization*, 1970, pp. 20–24.

[140] T. Elgamal, S. Luo, M. Boehm, A. V. Evfimievski, S. Tatikonda, B. Reinwald, and P. Sen, "Spoof: Sum-product optimization and operator fusion for large-scale machine learning." in *CIDR*, 2017.

[141] Q. Wang, M. Xu, C. Jin, X. Dong, J. Yuan, X. Jin, G. Huang, Y. Liu, and X. Liu, "Melon: Breaking the memory wall for resource-efficient on-device machine learning," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022, pp. 450–463.

[142] M. J. Fenske, E. Aminoff, N. Gronau, and M. Bar, "Top-down facilitation of visual object recognition: object-based and context-based contributions," *Progress in brain research*, vol. 155, pp. 3–21, 2006.

[143] W.-F. Lin, S. K. Reinhardt, and D. Burger, "Reducing dram latencies with an integrated memory hierarchy design," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*. IEEE, 2001, pp. 301–312.

[144] "Raspberry pi series processors," https://www.raspberrypi.com/products/.

[145] "Microcontrollers and microprocessors," https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors.

[146] S. Liu, X. Li, Z. Zhou, B. Guo, M. Zhang, H. Shen, and Z. Yu, "Adaenlight: Energy-aware low-light video stream enhancement on mobile devices," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 4, pp. 1–26, 2023.

[147] J.-W. Lai, "Opportunity and challenge of chiplet-based hpc and aiot," in *2021 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, 2021, pp. 1–2.

[148] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "Nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 81–93.

[149] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for fpga-based convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[150] N. K. Jha and S. Mittal, "Modeling data reuse in deep neural networks by taking data-types into cognizance," *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1526–1538, 2020.

[151] N. K. Jha, S. Mittal, and G. Mattela, "The ramifications of making deep neural networks compact," in *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*. IEEE, 2019, pp. 215–220.

[152] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[153] H. Wang, B. Guo, J. Liu, S. Liu, Y. Wu, and Z. Yu, "Context-aware adaptive surgery: A fast and effective framework for adaptative model partition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 3, pp. 1–22, 2021.

[154] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[155] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[156] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[157] J. Cheng, P.-s. Wang, G. Li, Q.-h. Hu, and H.-q. Lu, "Recent advances in efficient computation of deep convolutional neural networks," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, pp. 64–77, 2018.

[158] Y. Deng, "Deep learning on mobile devices: a review," in *Mobile Multimedia/Image Processing, Security, and Applications 2019*, vol. 10993. SPIE, 2019, pp. 52–66.

[159] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artificial Intelligence Review*, vol. 53, pp. 5113–5155, 2020.

[160] X. He, Z. Zhou, and L. Thiele, "Multi-task zipping via layer-wise neuron sharing," in *Advances in Neural Information Processing Systems*, 2018, pp. 6019–6029.

[161] D. Gao, X. He, Z. Zhou, Y. Tong, and L. Thiele, "Pruning meta-trained networks for on-device adaptation," in *Proceedings of the ACM International Conference on Information & Knowledge Management*, 2021, pp. 514–523.

[162] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *arXiv preprint arXiv:1507.06149*, 2015.

[163] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[164] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[165] C. Tai, T. Xiao, Y. Zhang, X. Wang *et al.*, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2015.

[166] S. Lin, R. Ji, C. Chen, D. Tao, and J. Luo, "Holistic cnn compression via low-rank decomposition with knowledge transfer," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 12, pp. 2889–2905, 2018.

[167] S. Mehta, M. Rastegari, L. Shapiro, and H. Hajishirzi, "Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 9190–9200.

[168] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2820–2828.

[169] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[170] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[171] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4820–4828.

[172] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[173] Z. Li, B. Ni, W. Zhang, X. Yang, and W. Gao, "Performance guaranteed network acceleration via high-order residual quantization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2584–2592.

[174] F. C.-T. Chow, *A portable machine-independent global optimizer–Design and measurements*. Stanford University, 1984.

[175] J. Fang, Y. Shen, Y. Wang, and L. Chen, "Optimizing dnn computation graph using graph substitutions," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2734–2746, 2020.

[176] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[177] X. Wei, Y. Liang, and J. Cong, "Overcoming data transfer bottlenecks in fpga-based dnn accelerators via layer conscious memory management," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[178] A. Symons, L. Mei, and M. Verhelst, "Loma: Fast auto-scheduling on dnn accelerators through loop-order-based memory allocation," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2021, pp. 1–4.

[179] C. Ji, Z. Zhu, X. Wang, W. Zhai, X. Zong, A. Chen, and M. Zhou, "Task-aware swapping for efficient dnn inference on dram-constrained edge systems," *International Journal of Intelligent Systems*, vol. 37, no. 10, pp. 8155–8169, 2022.

[180] G. S. Murthy, M. Ravishankar, M. M. Baskaran, and P. Sadayappan, "Optimal loop unrolling for gpgpu programs," in *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2010, pp. 1–11.

[181] H. Vanholder, "Efficient inference with tensorrt," in *GPU Technology Conference*, vol. 1, 2016, p. 2.

[182] X. Jiang, H. Wang, Y. Chen, Z. Wu, L. Wang, B. Zou, Y. Yang, Z. Cui, Y. Cai, T. Yu *et al.*, "Mnn: A universal and efficient inference engine," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 1–13, 2020.

[183] Z. Jia, O. Padon, J. Thomas, T. Warszawski, M. Zaharia, and A. Aiken, "Taso: optimizing deep learning computation with automatic generation of graph substitutions," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 47–62.

[184] T. Sekiyama, T. Imamichi, H. Imai, and R. Raymond, "Profile-guided memory optimization for deep neural networks," *arXiv preprint arXiv:1804.10001*, 2018.

[185] C.-C. Huang, G. Jin, and J. Li, "Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 1341–1355.

[186] P. S. Rawat, A. Sukumaran-Rajam, A. Rountev, F. Rastello, L.-N. Pouchet, and P. Sadayappan, "Associative instruction reordering to alleviate register pressure," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 590–602.

[187] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Magnet: A modular accelerator generator for neural networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[188] Y. Shen, M. Ferdman, and P. Milder, "Maximizing cnn accelerator efficiency through resource partitioning," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 535–547, 2017.

[189] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

[190] A. Stoutchinin, F. Conti, and L. Benini, "Optimally scheduling cnn convolutions for efficient memory access," *arXiv preprint arXiv:1902.01492*, 2019.

[191] X. Peng, X. Shi, H. Dai, H. Jin, W. Ma, Q. Xiong, F. Yang, and X. Qian, "Capuchin: Tensor-based gpu memory management for deep learning," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 891–905.

[192] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5687–5695.

[193] J. Farley and A. Gerstlauer, "Memory-aware fusing and tiling of neural networks for accelerated edge inference," *arXiv preprint arXiv:2107.06960*, 2021.

[194] R. Stahl, Z. Zhao, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Fully distributed deep learning inference on resource-constrained edge devices," in *International Conference on Embedded Computer Systems*. Springer, 2019, pp. 77–90.

[195] S. Naveen and M. R. Kounte, "Memory optimization at edge for distributed convolution neural network," *Transactions on Emerging Telecommunications Technologies*, vol. 33, no. 12, p. e4648, 2022.

[196] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things," *IEEE Network*, vol. 33, no. 5, pp. 96–103, 2019.

[197] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iraf: A deep reinforcement learning approach for collaborative mobile edge computing iot networks," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 7011–7024, 2019.

[198] S. Zhang, Y. Li, X. Liu, S. Guo, W. Wang, J. Wang, B. Ding, and D. Wu, "Towards real-time cooperative deep inference over the cloud and edge end devices," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 2, pp. 1–24, 2020.

[199] G. Pan, H. Zhang, S. Xu, S. Zhang, and X. Chen, "Joint optimization of dnn inference delay and energy under accuracy constraints for ar applications," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 2230–2235.

[200] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware dnn inference in edge computing by exploring dnn model partitioning and inference parallelism," *IEEE Transactions on Mobile Computing*, 2021.

[201] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "Ionn: Incremental offloading of neural network computations from mobile devices to edge

servers," in *Proceedings of the ACM symposium on cloud computing*, 2018, pp. 401–411.
[202] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2019.
[203] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing the convolution operation to accelerate deep neural networks on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1354–1367, 2018.
[204] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "Spinn: synergistic progressive inference of neural networks over device and cloud," in *Proceedings of the 26th annual international conference on mobile computing and networking*, 2020, pp. 1–15.
[205] F. Zhang, J. Fang, B. Wah, and P. Torr, "Deep fusionnet for point cloud semantic segmentation," in *European Conference on Computer Vision*. Springer, 2020, pp. 644–663.
[206] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, 2018, pp. 31–36.
[207] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.
[208] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.
[209] L. Yang, A. S. Rakin, and D. Fan, "Rep-net: Efficient on-device learning via feature reprogramming," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 277–12 286.
[210] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.
[211] A. L. Friesen and P. Domingos, "Deep learning as a mixed convex-combinatorial optimization problem," *arXiv preprint arXiv:1710.11573*, 2017.
[212] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Towards effective low-bitwidth convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7920–7928.
[213] A. Finkelstein, U. Almog, and M. Grobman, "Fighting quantization bias with bias," *arXiv preprint arXiv:1906.03193*, 2019.
[214] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.
[215] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.
[216] G. Wang, Z. Liu, Z. Jiang, N. Liu, N. Zou, and X. Hu, "Towards memory efficient training via dual activation precision," *arXiv preprint arXiv:2208.04187*, 2022.
[217] J. Yu, A. Lukefahr, R. Das, and S. Mahlke, "Tf-net: Deploying sub-byte deep neural networks on microcontrollers," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–21, 2019.
[218] Q. Lu, W. Jiang, X. Xu, J. Hu, and Y. Shi, "Quantization through search: A novel scheme to quantize convolutional neural networks in finite weight space," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 378–383.
[219] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[220] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
[221] G. Kaplun, A. Gurevich, T. Swisa, M. David, S. Shalev-Shwartz, and E. Malach, "Subtuning: Efficient finetuning for multi-task learning," *arXiv preprint arXiv:2302.06354*, 2023.
[222] A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves, "Memory-efficient backpropagation through time," *Advances in neural information processing systems*, vol. 29, 2016.
[223] M. Kirisame, S. Lyubomirsky, A. Haan, J. Brennan, M. He, J. Roesch, T. Chen, and Z. Tatlock, "Dynamic tensor rematerialization," *arXiv preprint arXiv:2006.09616*, 2020.
[224] I. Gim and J. Ko, "Memory-efficient dnn training on mobile devices," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022, pp. 464–476.
[225] R. D. Evans, L. Liu, and T. M. Aamodt, "Jpeg-act: accelerating deep learning via transform-based lossy compression," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 860–873.
[226] A. Hosny, M. Neseem, and S. Reda, "Sparse bitmap compression for memory-efficient training on the edge," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2021, pp. 14–25.
[227] C. Unger, Z. Jia, W. Wu, S. Lin, M. Baines, C. E. Q. Narvaez, V. Ramakrishnaiah, N. Prajapati, P. McCormick, J. Mohd-Yusof *et al.*, "Unity: Accelerating {DNN} training through joint optimization of algebraic transformations and parallelization," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 267–284.
[228] S.-M. Hu, D. Liang, G.-Y. Yang, G.-W. Yang, and W.-Y. Zhou, "Jittor: a novel deep learning framework with meta-operators and unified graph execution," *Science China Information Sciences*, vol. 63, pp. 1–21, 2020.
[229] Z. Zheng, P. Zhao, G. Long, F. Zhu, K. Zhu, W. Zhao, L. Diao, J. Yang, and W. Lin, "Fusionstitching: boosting memory intensive computations for deep learning workloads," *arXiv preprint arXiv:2009.10924*, 2020.
[230] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, "Data movement is all you need: A case study on optimizing transformers," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 711–732, 2021.
[231] Z. Zheng, X. Yang, P. Zhao, G. Long, K. Zhu, F. Zhu, W. Zhao, X. Liu, J. Yang, J. Zhai *et al.*, "Astitch: enabling a new multi-dimensional optimization space for memory-intensive ml training and inference on modern simt architectures," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 359–373.
[232] X. Nie, X. Miao, Z. Yang, and B. Cui, "Tsplit: Fine-grained gpu memory management for efficient dnn training via tensor splitting," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 2615–2628.
[233] T. D. Le, H. Imai, Y. Negishi, and K. Kawachiya, "Automatic gpu memory management for large neural models in tensorflow," in *Proceedings of the 2019 ACM SIGPLAN International Symposium on Memory Management*, 2019, pp. 1–13.
[234] S. Shriram, A. Garg, and P. Kulkarni, "Dynamic memory management for gpu-based training of deep neural networks," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 200–209.
[235] J. Ren, J. Luo, K. Wu, M. Zhang, H. Jeon, and D. Li, "Sentinel: Efficient tensor migration and allocation on heterogeneous memory systems for deep learning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 598–611.
[236] P. Chen, S. He, X. Zhang, S. Chen, P. Hong, Y. Yin, X.-H. Sun, and G. Chen, "Cswap: A self-tuning compression framework for accelerating tensor swapping in gpus," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021, pp. 271–282.
[237] J. Li, X. Wang, X. Chen, G. Li, X. Dong, P. Zhao, X. Yu, Y. Yang, W. Cao, L. Liu *et al.*, "An application-oblivious memory scheduling system for dnn accelerators," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 19, no. 4, pp. 1–26, 2022.
[238] J. Jung, J. Kim, and J. Lee, "Deepum: Tensor migration and prefetching in unified memory," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 207–221. [Online]. Available: https://doi.org/10.1145/3575693.3575736
[239] J. Roesch, S. Lyubomirsky, M. Kirisame, L. Weber, J. Pollock, L. Vega, Z. Jiang, T. Chen, T. Moreau, and Z. Tatlock, "Relay: A high-level compiler for deep learning," *arXiv preprint arXiv:1904.08368*, 2019.
[240] Tencent, "Ncnn," https://github.com/Tencent/ncnn/.
[241] Huawei, "Mindspore," https://github.com/mindspore-ai/mindspore.
[242] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean, "Hierarchical planning for device placement," 2018.
[243] Y. Gao, L. Chen, and B. Li, "Spotlight: Optimizing device placement for training deep neural networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1676–1684.
[244] R. Addanki, S. B. Venkatakrishnan, S. Gupta, H. Mao, and M. Alizadeh, "Placeto: Learning generalizable device placement algorithms

for distributed machine learning," *arXiv preprint arXiv:1906.08879*, 2019.
[245] A. Bortfeldt, "A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces," *European Journal of Operational Research*, vol. 172, no. 3, pp. 814–837, 2006.
[246] "Windows virtual address spaces," https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/virtual-address-spaces.
[247] https://www.kernel.org/doc/html/latest/index.html#, title = The Linux Kernel documentation,.
[248] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.
[249] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.
[250] D. Narayanan, A. Phanishayee, K. Shi, X. Chen, and M. Zaharia, "Memory-efficient pipeline-parallel dnn training," in *Proceedings of ICML*. PMLR, 2021, pp. 7937–7947.
[251] Y. Liu, S. Li, J. Fang, Y. Shao, B. Yao, and Y. You, "Map: Memory-aware automated intra-op parallel training for foundation models," *arXiv preprint arXiv:2302.02599*, 2023.
[252] R. Mcdonald, M. Mohri, N. Silberman, D. Walker, and G. Mann, "Efficient large-scale distributed training of conditional maximum entropy models," *Advances in neural information processing systems*, vol. 22, 2009.
[253] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, "A reliable effective terascale linear learning system," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1111–1133, 2014.
[254] J. Wei, W. Dai, A. Qiao, Q. Ho, H. Cui, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Managed communication and consistency for fast data-parallel iterative analytics," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, 2015, pp. 381–394.
[255] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, "Parameter server for distributed machine learning," in *Big learning NIPS workshop*, vol. 6, no. 2, 2013.
[256] O. Beaumont, L. Eyraud-Dubois, and A. Shilova, "Madpipe: Memory aware dynamic programming algorithm for pipelined model parallelism," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2022, pp. 1063–1073.
[257] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.
[258] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.
[259] X. Ouyang, Z. Xie, J. Zhou, J. Huang, and G. Xing, "Clusterfl: a similarity-aware federated learning system for human activity recognition," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 54–66.
[260] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.
[261] V. Md, S. Misra, G. Ma, R. Mohanty, E. Georganas, A. Heinecke, D. Kalamkar, N. K. Ahmed, and S. Avancha, "Distgnn: Scalable distributed training for large-scale graph neural networks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–14.
[262] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.
[263] C. Rosset, "Turing-nlg: A 17-billion-parameter language model by microsoft," *Microsoft Blog*, vol. 1, no. 2, 2020.
[264] A. Heinecke, G. Henry, M. Hutchinson, and H. Pabst, "Libxsmm: accelerating small matrix multiplications by runtime code generation," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 981–991.
[265] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient federated learning via guided participant selection." in *OSDI*, 2021, pp. 19–35.
[266] J. Sun, A. Li, L. Duan, S. Alam, X. Deng, X. Guo, H. Wang, M. Gorlatova, M. Zhang, H. Li *et al.*, "Fedsea: A semi-asynchronous federated learning framework for extremely heterogeneous devices," 2022.
[267] N. Band, "Memflow: Memory-aware distributed deep learning," in *Proceedings of ACM SIGMOD*, 2020, pp. 2883–2885.
[268] O. Beaumont, J. Herrmann, G. Pallez, and A. Shilova, "Optimal memory-aware backpropagation of deep join networks," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20190049, 2020.
[269] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.
[270] M. E. Paoletti, J. M. Haut, X. Tao, J. Plaza, and A. Plaza, "Flop-reduction through memory allocations within cnn for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 7, pp. 5938–5952, 2020.
[271] O. Saha, A. Kusupati, H. V. Simhadri, M. Varma, and P. Jain, "Rnnpool: efficient non-linear pooling for ram constrained inference," *Advances in Neural Information Processing Systems*, vol. 33, pp. 20 473–20 484, 2020.
[272] S. Oh, J. Choi, J.-K. Kim, and J. Kim, "Quantum convolutional neural network for resource-efficient image classification: A quantum random access memory (qram) approach," in *2021 International Conference on Information Networking (ICOIN)*. IEEE, 2021, pp. 50–52.
[273] D. Liang, J. Shiomi, N. Miura, and H. Awano, "Distrihd: A memory efficient distributed binary hyperdimensional computing architecture for image classification," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 43–49.
[274] T. Emara, H. E. Abd El Munim, and H. M. Abbas, "Liteseg: A novel lightweight convnet for semantic segmentation," in *2019 Digital Image Computing: Techniques and Applications (DICTA)*. IEEE, 2019, pp. 1–7.
[275] Y. Wang, Q. Zhou, J. Liu, J. Xiong, G. Gao, X. Wu, and L. J. Latecki, "Lednet: A lightweight encoder-decoder network for real-time semantic segmentation," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 1860–1864.
[276] I. Krešo, J. Krapac, and S. Šegvić, "Efficient ladder-style densenets for semantic segmentation of large images," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 4951–4961, 2020.
[277] Y. Jin, D. Han, and H. Ko, "Memory-based semantic segmentation for off-road unstructured natural environments," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 24–31.
[278] Y. Shangguan, J. Li, Q. Liang, R. Alvarez, and I. McGraw, "Optimizing speech recognition for the edge," *arXiv preprint arXiv:1909.12408*, 2019.
[279] J. Guo, G. Tiwari, J. Droppo, M. Van Segbroeck, C.-W. Huang, A. Stolcke, and R. Maas, "Efficient minimum word error rate training of rnn-transducer for end-to-end speech recognition," *arXiv preprint arXiv:2007.13802*, 2020.
[280] G. I. Winata, S. Cahyawijaya, Z. Lin, Z. Liu, and P. Fung, "Lightweight and efficient end-to-end speech recognition using low-rank transformer," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6144–6148.
[281] Q. Wang, I. L. Moreno, M. Saglam, K. Wilson, A. Chiao, R. Liu, Y. He, W. Li, J. Pelecanos, M. Nika *et al.*, "Voicefilter-lite: Streaming targeted voice separation for on-device speech recognition," *arXiv preprint arXiv:2009.04323*, 2020.
[282] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.
[283] S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi, and J. A. Benediktsson, "Deep learning for hyperspectral image classification: An overview," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 9, pp. 6690–6709, 2019.
[284] Z. Weng and Z. Qin, "Semantic communication systems for speech transmission," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2434–2444, 2021.
[285] N. Das, S. Chakraborty, J. Chaki, N. Padhy, and N. Dey, "Fundamentals, present and future perspectives of speech enhancement," *International Journal of Speech Technology*, vol. 24, pp. 883–901, 2021.
[286] B. Zhao, J. Feng, X. Wu, and S. Yan, "A survey on deep learning-based fine-grained object classification and semantic segmentation," *International Journal of Automation and Computing*, vol. 14, no. 2, pp. 119–135, 2017.
[287] P. Gupta, N. Saxena, M. Sharma, and J. Tripathi, "Deep neural network for human face recognition," *International Journal of Engineering and Manufacturing (IJEM)*, vol. 8, no. 1, pp. 63–71, 2018.

[288] H. You, S. Tian, L. Yu, and Y. Lv, "Pixel-level remote sensing image recognition based on bidirectional word vectors," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 2, pp. 1281–1293, 2019.

[289] M. Protasov, G. Reshetova, and V. Tcheverda, "Fracture detection by gaussian beam imaging of seismic data and image spectrum analysis," *Geophysical prospecting*, vol. 64, no. 1, pp. 68–82, 2016.

[290] X. Qian, R. Hang, and Q. Liu, "Rex: an efficient approach to reducing memory cost in image classification," 2022.

[291] W. Sun and R. Wang, "Fully convolutional networks for semantic segmentation of very high resolution remotely sensed images combined with dsm," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 3, pp. 474–478, 2018.

[292] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *Journal of cleaner production*, vol. 140, pp. 1454–1464, 2017.

[293] M. Alaa, A. A. Zaidan, B. B. Zaidan, M. Talal, and M. L. M. Kiah, "A review of smart home applications based on internet of things," *Journal of Network and Computer Applications*, vol. 97, pp. 48–65, 2017.

[294] S. Feng, X. Yan, H. Sun, Y. Feng, and H. X. Liu, "Intelligent driving intelligence test for autonomous vehicles with naturalistic and adversarial environment," *Nature communications*, vol. 12, no. 1, p. 748, 2021.

[295] Q.-T.-A. Khan, S. Abbas, M. A. Khan, A. Fatima, S. Alanazi, and N. S. Elmitwally, "Modelling intelligent driving behaviour using machine learning," 2021.

[296] T. Yoshioka, H. Erdogan, Z. Chen, and F. Alleva, "Multi-microphone neural speech separation for far-field multi-talker speech recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5739–5743.

[297] V. Pratap, Q. Xu, J. Kahn, G. Avidov, T. Likhomanenko, A. Hannun, V. Liptchinsky, G. Synnaeve, and R. Collobert, "Scaling up online speech recognition using convnets," *arXiv preprint arXiv:2001.09727*, 2020.

[298] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 75–84.

[299] D. Liciotti, M. Bernardini, L. Romeo, and E. Frontoni, "A sequential deep learning application for recognising human activities in smart homes," *Neurocomputing*, vol. 396, pp. 501–513, 2020.

[300] S. Mekruksavanich and A. Jitpattanakul, "Lstm networks using smartphone data for sensor-based human activity recognition in smart homes," *Sensors*, vol. 21, no. 5, p. 1636, 2021.

[301] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "wav2vec: Unsupervised pre-training for speech recognition," *arXiv preprint arXiv:1904.05862*, 2019.

[302] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," *arXiv preprint arXiv:2212.04356*, 2022.

[303] Y. Liu, J. James, J. Kang, D. Niyato, and S. Zhang, "Privacy-preserving traffic flow prediction: A federated learning approach," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7751–7763, 2020.

[304] E. J. Hoffmann, Y. Wang, M. Werner, J. Kang, and X. X. Zhu, "Model fusion for building type classification from aerial and street view images," *Remote Sensing*, vol. 11, no. 11, p. 1259, 2019.

[305] X. Yi, J. Zhang, Z. Wang, T. Li, and Y. Zheng, "Deep distributed fusion network for air quality prediction," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 965–973.

[306] G. Manogaran, M. Alazab, P. M. Shakeel, and C.-H. Hsu, "Blockchain assisted secure data sharing model for internet of things based smart industries," *IEEE Transactions on Reliability*, vol. 71, no. 1, pp. 348–358, 2021.

[307] Y. Shahzad, H. Javed, H. Farman, J. Ahmad, B. Jan, and M. Zubair, "Internet of energy: Opportunities, applications, architectures and challenges in smart industries," *Computers & Electrical Engineering*, vol. 86, p. 106739, 2020.

[308] M. Andronie, G. Lăzăroiu, M. Iatagan, C. Uță, R. Ștefănescu, and M. Cocoșatu, "Artificial intelligence-based decision-making algorithms, internet of things sensing networks, and deep learning-assisted smart process management in cyber-physical production systems," *Electronics*, vol. 10, no. 20, p. 2497, 2021.

[309] T. Liebig, N. Piatkowski, C. Bockermann, and K. Morik, "Dynamic route planning with real-time traffic predictions," *Information Systems*, vol. 64, pp. 258–265, 2017.

[310] N. G. Polson and V. O. Sokolov, "Deep learning for short-term traffic flow prediction," *Transportation Research Part C: Emerging Technologies*, vol. 79, pp. 1–17, 2017.

[311] H. Ren, D. Anicic, and T. Runkler, "How to manage tiny machine learning at scale: An industrial perspective," *arXiv preprint arXiv:2202.09113*, 2022.

[312] K. Yang, Z. Yu, and Y. Luo, "Analysis on driving factors of lake surface water temperature for major lakes in yunnan-guizhou plateau," *Water Research*, vol. 184, p. 116018, 2020.

[313] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 517–532, 2021.

[314] B. Sudharsan, J. G. Breslin, and M. I. Ali, "Rce-nn: a five-stage pipeline to execute neural networks (cnns) on resource-constrained iot edge devices," in *Proceedings of the 10th International Conference on the Internet of Things*, 2020, pp. 1–8.

[315] A. Maskey and M. Cho, "Cubesatnet: Ultralight convolutional neural network designed for on-orbit binary image classification on a 1u cubesat," *Engineering Applications of Artificial Intelligence*, vol. 96, p. 103952, 2020.

[316] C.-W. Hung, S.-X. Zeng, C.-H. Lee, and W.-T. Li, "End-to-end deep learning by mcu implementation: an intelligent gripper for shape identification," *Sensors*, vol. 21, no. 3, p. 891, 2021.

[317] Y. Wu, Z. Wang, Y. Shi, and J. Hu, "Enabling on-device cnn training by self-supervised instance filtering and error map pruning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3445–3457, 2020.

[318] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[319] Q. Wang, D. Li, X. Huang, S. Shen, S. Mei, and J. Liu, "Optimizing fft-based convolution on armv8 multi-core cpus," in *European Conference on Parallel Processing*. Springer, 2020, pp. 248–262.

[320] J. Meng, C. Zhuang, P. Chen, M. Wahib, B. Schmidt, X. Wang, H. Lan, D. Wu, M. Deng, Y. Wei *et al.*, "Automatic generation of high-performance convolution kernels on arm cpus for deep learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2885–2899, 2022.

[321] D. Li, D. Huang, Z. Chen, and Y. Lu, "Optimizing massively parallel winograd convolution on arm processor," in *50th International Conference on Parallel Processing*, 2021, pp. 1–12.

[322] X. Huang, Q. Wang, S. Lu, R. Hao, S. Mei, and J. Liu, "Numa-aware fft-based convolution on armv8 many-core cpus," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2021, pp. 1019–1026.

[323] X. Zhou, Y. Dou, R. Li, P. Zhang, and Y. Liu, "A pipelining strategy for accelerating convolution neural networks on arm cpus," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 2, p. e6102, 2022.

[324] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "Dlau: A scalable deep learning accelerator unit on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 513–517, 2016.

[325] X. Xie and C. Wu, "Wpu: A fpga-based scalable, efficient and software/hardware co-design deep neural network inference acceleration processor," in *2021 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*. IEEE, 2021, pp. 1–5.

[326] J. Meng, S. K. Venkataramanaiah, C. Zhou, P. Hansen, P. Whatmough, and J.-s. Seo, "Fixyfpga: Efficient fpga accelerator for deep neural networks with high element-wise sparsity and without external memory access," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2021, pp. 9–16.

[327] X. Chang, H. Pan, D. Zhang, Q. Sun, and W. Lin, "A memory-optimized and energy-efficient cnn acceleration architecture based on fpga," in *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*. IEEE, 2019, pp. 2137–2141.

[328] A. Rios-Navarro, R. Tapiador-Morales, A. Jimenez-Fernandez, C. Amaya, M. Dominguez-Morales, T. Delbruck, and A. Linares-Barranco, "Performance evaluation over hw/sw co-design soc memory transfers for a cnn accelerator," in *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)*. IEEE, 2018, pp. 1–4.

[329] H. Fu, Z. Niu, C. Zhang, J. Ma, and J. Chen, "Visual cortex inspired cnn model for feature construction in text analysis," *Frontiers in computational neuroscience*, vol. 10, p. 64, 2016.

[330] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra, "High-throughput cnn inference on embedded arm big. little multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2254–2267, 2019.

[331] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, and A. Rabinovich, "Going deeper with convolutions," *IEEE Computer Society*, 2014.

[332] P. Liu, W. Qingqing, and W. Liu, "Enterprise human resource management platform based on fpga and data mining," *Microprocessors and Microsystems*, vol. 80, p. 103330, 2021.

[333] H. Peng, S. Zhou, S. Weitze, J. Li, S. Islam, T. Geng, A. Li, W. Zhang, M. Song, M. Xie *et al.*, "Binary complex neural network acceleration on fpga," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2021, pp. 85–92.

[334] G. Korol and F. G. Moraes, "A fpga parameterizable multi-layer architecture for cnns," in *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design*, 2019, pp. 1–6.

[335] M. Zainab, A. R. Usmani, S. Mehrban, and M. Hussain, "Fpga based implementations of rnn and cnn: A brief analysis," in *2019 International Conference on Innovative Computing (ICIC)*. IEEE, 2019, pp. 1–8.

[336] L. Petrica, T. Alonso, M. Kroes, N. Fraser, S. Cotofana, and M. Blott, "Memory-efficient dataflow inference for deep cnns on fpga," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 48–55.

[337] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina *et al.*, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 369–383.

[338] K. Udagawa, Y. Saito, and H. Saruwatari, "Human-in-the-loop speaker adaptation for dnn-based multi-speaker tts," *arXiv preprint arXiv:2206.10256*, 2022.

[339] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[340] Y. Li, C. Hao, X. Zhang, X. Liu, Y. Chen, J. Xiong, W.-m. Hwu, and D. Chen, "Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[341] W. Jiang, X. Zhang, E. H.-M. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[342] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "Fann-on-mcu: An open-source toolkit for energy-efficient neural network inference at the edge of the internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4403–4417, 2020.

[343] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "Pulp-nn: accelerating quantized neural networks on parallel ultra-low-power risc-v processors," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, p. 20190155, 2020.

[344] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 907–922.

[345] S. Liu, B. Ren, X. Shen, and Y. Wang, "Cocopie: Making mobile ai sweet as pie–compression-compilation co-design goes a long way," *arXiv preprint arXiv:2003.06700*, 2020.

[346] X. Ma, F.-M. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang, "Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5117–5124.

[347] "Nvidia jetson embedded systems," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.

[348] "Xilinx zynq-7000 fpgas," https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[349] D. Baymurzina, E. Golikov, and M. Burtsev, "A review of neural architecture search," *Neurocomputing*, vol. 474, pp. 82–93, 2022.

[350] Y. Kim, Y. Li, H. Park, Y. Venkatesha, and P. Panda, "Neural architecture search for spiking neural networks," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV*. Springer, 2022, pp. 36–56.

[351] X. Liu, J. Zhao, J. Li, B. Cao, and Z. Lv, "Federated neural architecture search for medical data security," *IEEE transactions on industrial informatics*, vol. 18, no. 8, pp. 5628–5636, 2022.

[352] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," *arXiv preprint arXiv:1902.08142*, 2019.

[353] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in artificial intelligence*. PMLR, 2020, pp. 367–377.

[354] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, "Edge computing with artificial intelligence: A machine learning perspective," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[355] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 10, pp. 4229–4238, 2019.

[356] Y. Huang, L. Chu, Z. Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang, "Personalized cross-silo federated learning on non-iid data," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, 2021, pp. 7865–7873.

[357] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, N. Karianakis, Y. Shu, K. Hsieh, V. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *USENIX NSDI*, 2022.

[358] R. T. Mullapudi, S. Chen, K. Zhang, D. Ramanan, and K. Fatahalian, "Online model distillation for efficient video inference," in *Proceedings of the IEEE/CVF International conference on computer vision*, 2019, pp. 3573–3582.

[359] M. Khani, P. Hamadanian, A. Nasr-Esfahany, and M. Alizadeh, "Real-time video inference on edge devices via adaptive model streaming," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 4572–4582.

[360] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[361] M. Wordeman, J. Silberman, G. Maier, and M. Scheuermann, "A 3d system prototype of an edram cache stacked over processor-like logic using through-silicon vias," in *2012 IEEE International Solid-State Circuits Conference*. IEEE, 2012, pp. 186–187.

[362] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, and F. Franchetti, "A 3d-stacked logic-in-memory accelerator for application-specific data intensive computing," in *2013 IEEE international 3D systems integration conference (3DIC)*. IEEE, 2013, pp. 1–7.

[363] Q. Zhu, T. Graf, H. E. Sumbul, L. Pileggi, and F. Franchetti, "Accelerating sparse matrix-matrix multiplication with 3d-stacked logic-in-memory hardware," in *2013 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2013, pp. 1–6.

[364] V. Seshadri, K. Hsieh, A. Boroum, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast bulk bitwise and and or in dram," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 127–131, 2015.

[365] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch *et al.*, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 185–197.

[366] N. E. Jerger, L.-S. Peh, and M. Lipasti, "Virtual circuit tree multicasting: A case for on-chip hardware multicast support," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 229–240, 2008.

[367] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, "Flexram: Toward an advanced intelligent memory system," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*. IEEE, 2012, pp. 5–14.

[368] Y. Wang, T. Tang, L. Xia, B. Li, P. Gu, H. Yang, H. Li, and Y. Xie, "Energy efficient rram spiking neural network for real time classification," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, 2015, pp. 189–194.

[369] C. Liu, B. Yan, C. Yang, L. Song, Z. Li, B. Liu, Y. Chen, H. Li, Q. Wu, and H. Jiang, "A spiking neuromorphic design with resistive crossbar," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.