electronics

MDPI

Article

Optimization Methods, Challenges, and Opportunities for Edge Inference: A Comprehensive Survey

Runhua Zhang, Hongxu Jiang *, Wei Wang and Jinhao Liu

School of Computer Science and Engineering, Beihang University, Beijing 100191, China; rhzhang20@buaa.edu.cn (R.Z.); ww_bh@buaa.edu.cn (W.W.); robertliu@buaa.edu.cn (J.L.)
* Correspondence: jianghx@buaa.edu.cn

Abstract: Artificial intelligence (AI) continues to enhance production efficiency across various fields of society. Considering real-time requirements and privacy issues, edge inference (EI) is shifting from cloud scenarios to edge scenarios. As intelligent models grow in complexity and size, EI encounters significant challenges. To address these, existing research works have optimized EI from four aspects (model design, model compression, compilation toolchain, and collaborative inference) to ensure the advantages of edge intelligence. However, current works lack a comprehensive classification and discussion of existing research results. Thus, we conduct a comprehensive survey on their state-of-the-art research. Specifically, we first review the background and motivation of EI, then analyze the key issues, characteristics, and technologies of each direction. Finally, we analyze future development trends. This paper can help researchers quickly sort out the different directions of EI optimization and important related work. We hope it can bring inspiration to the researchers in these communities and motivate more follow-up works.

Keywords: edge inference; model design; model compression; compilation toolchain; collaborative inference

Academic Editor: Manuel Mazzara

Received: 2 March 2025
Revised: 17 March 2025
Accepted: 25 March 2025
Published: 27 March 2025

Citation: Zhang, R.; Jiang, H.; Wang, W.; Liu, J. Optimization Methods, Challenges, and Opportunities for Edge Inference: A Comprehensive Survey. *Electronics* **2025**, *14*, 1345. https://doi.org/10.3390/electronics14071345

1. Introduction

The rapid development of deep neural networks (DNNs) has provided strong support for various intelligent applications, such as autonomous driving [1–3], medical diagnosis [4–6], smart homes [7–9], smart cities [10–12], etc. The development and application of DNNs can be divided into two important stages: model training and model inference. Model training involves utilizing the combined computational power of multiple servers or server clusters to perform data analysis and learning, ultimately producing a trained model. Model inference typically refers to using the trained model to analyze and make predictions based on data collected from various physical sensors. With the rise of the Internet of things (IoT), the number of edge hardware devices is increasing rapidly, and they have gradually become an important carrier for DNN applications. Edge hardware has the inherent advantage of being able to directly connect to various physical sensors to achieve rapid data acquisition, which can effectively avoid communication latency [13,14]. Importantly, this direct connectivity also plays a crucial role in enhancing security and protecting privacy [15–17].

Compared with cloud-based devices, edge hardware has many advantages, such as small size, low power consumption, strong flexibility, and high reliability. However, edge hardware also has obvious disadvantages. As the scale of the model continues to grow, the continuous growth of computing power requirements and the increasing complexity of

model structures have led to unprecedented challenges for edge inference (EI). Edge hardware is resource constrained, leading to poor real-time performance when deploying large DNNs. For example, related work [18] compares the inference times between edge devices (such as FPGAs) and server devices (such as GPUs). The inference on the edge device is optimized, while the inference on the server device is not optimized. Even so, the inference time on the edge device is still longer than that on the server device. Therefore, a large number of optimization techniques are used to accelerate EI efficiency (such as lightweight model design, model compression, compilation optimization, etc.). In lightweight model design, through the search for neural network architecture [19,20] or human experience design [21,22], we can design DNNs that are more suitable for the characteristics of edge hardware. Lightweight model design needs to consider how to maintain high accuracy and robustness of DNNs while significantly reducing computing resources and storage requirements. As for model compression, many optimization methods (such as model pruning [23,24], quantization acceleration [25,26], sparse optimization [27,28], etc.) can compress the computational complexity and parameter quantity of DNNs by continuously balancing inference accuracy and inference efficiency. It is necessary to overcome the problem of accuracy loss that may be caused during the compression process. Compilation optimization performs computational graph optimization [29,30] and operator optimization [31,32] for the given model, allowing the model to fully utilize the computing and storage advantages of edge devices. Compilation optimization focuses on improving model running efficiency. Its challenges include efficient code generation and scheduling for specific hardware architectures, and minimizing energy consumption and latency while meeting performance requirements. Meanwhile, it is also worth considering in more detail how these optimization methods adapt to different hardware platforms, such as GPU, ARM, and FPGA.

Although the above optimization methods can improve the inference efficiency, the resource-constrained characteristics of edge devices limit real-time performance. In recent years, various applications have adopted collaborative inference [33–35] schemes to alleviate this problem. Collaborative inference can divide the inference task into different sub-tasks, and then dynamically assign these sub-tasks to different nodes based on the characteristics of each node. Compared with the single-node scenario, collaborative inference can adopt flexible scheduling strategies and resource allocation methods to achieve lower inference latency or energy consumption. Intelligent collaborative inference involves collaboration between multiple models or devices. The core challenge is how to effectively allocate tasks, protect data privacy, and ensure real-time responsiveness and inference accuracy.

In summary, the above optimization methods have jointly promoted the development and application of EI. However, previous studies have tended to focus on specific aspects, failing to fully cover or integrate the diverse research paths and potential development directions within the field of edge inference. Furthermore, as technology advances and new challenges emerge, earlier surveys may not incorporate the latest research trends and needs in a timely manner, leading to inadequate discussion of current issues. Therefore, this paper conducts a comprehensive study of EI-related research works in recent years. Specifically, we first review the background of EI and classify the optimization directions. Next, we further summarize their optimization goals and effective techniques. Finally, we discuss several future development trends of EI.

2. Preliminaries

2.1. Introduction to Edge Inference

EI refers to the process of executing the DNN inference on edge devices at the source of data or close to the source (as shown in Figure 1). EI aims to reduce data communication latency, enhance real-time response capabilities, and improve data processing efficiency. By transferring computing tasks from the cloud to the edge, it can quickly process information locally and effectively solve bandwidth bottlenecks and privacy and security issues. It is particularly suitable for application scenarios that are sensitive to latency and require high privacy protection, such as autonomous driving, intelligent monitoring, and industrial automation. The primary distinction between edge inference and traditional cloud inference lies in the location of data processing. Edge inference performs data processing on local devices, offering reduced latency and enhanced security. However, due to the limitations of edge device computing resources, EI may combine certain cloud computing resources to form a collaborative inference model. In contrast, traditional cloud inference offloads all inference tasks to remote data centers. EI is becoming a key foundation for promoting innovation in these fields. In addition, with the continuous improvement and optimization of edge hardware performance, EI has shown great potential. It not only supports more complex model inference but also ensures efficient model performance and accuracy in resource-constrained environments.

However, edge hardware has limited resources compared to cloud computing centers, which poses many challenges for edge hardware in terms of adapting to growing computing power requirements and complex model structures. According to OpenAI's report [36], since the advent of AlexNet [37] in 2013, the computing power requirements of the model have doubled on average every 3.4 months, far exceeding the two-year cycle of Moore's Law. As of AlphaGoZero in 2018, the computing power requirements have increased by 300,000 times. Subsequently, pre-trained large models appeared, and the number of model parameters and computing requirements exploded by five orders of magnitude in five years, an average of 240 times per year. This not only puts higher requirements on model training but also brings unprecedented challenges to the real-time performance of model inference. On the one hand, existing work uses techniques such as neural network compression (such as pruning and quantization) to address these challenges. On the other hand, many fields have begun to explore collaboration-based solutions to alleviate performance bottlenecks on single edge hardware devices. The collaborative inference architecture needs to solve problems such as how to dynamically and evenly distribute computing tasks, minimize communication costs, and quickly generate optimal deployment solutions. Ultimately, efficient model deployment can be achieved to meet real-time requirements.

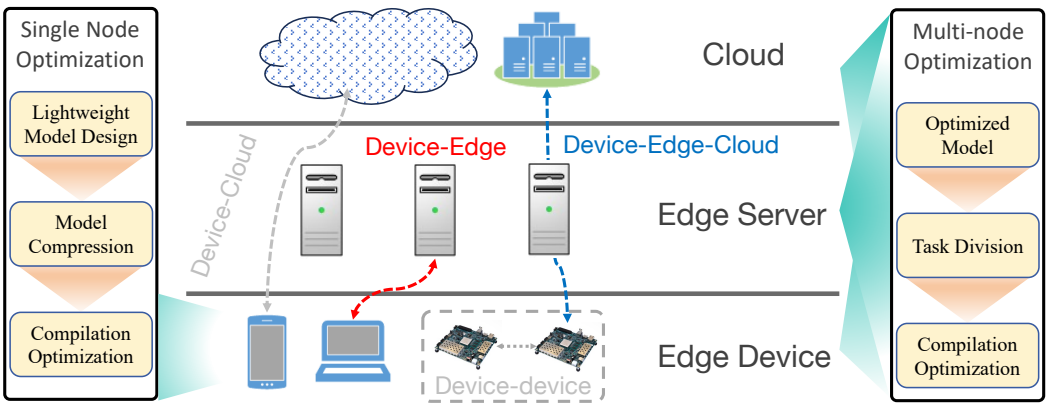


Figure 1. An illustration of the existing Edge Inference scenarios.

2.2. Optimizing Methods for EI

Considering the growing model size, diverse model structures, hardware resource constraints, and diverse task requirements. Various optimization methods have been proposed in different directions both domestically and internationally. Overall, this article summarizes them into four types:

(1) Lightweight model design. Lightweight model design usually uses depthwise separable convolution, pointwise convolution, and grouped convolution to replace traditional convolution layers, thereby reducing the number of parameters and computational complexity. In addition, carefully designed structures (such as MobileNet [21], SqueezeNet [22], and EfficientNet [38]) can achieve efficient information processing and feature extraction. They focus on minimizing the overhead of computing and storage resources while maintaining performance. Since manual design relies on a lot of expert experience, NAS technology searches for lightweight model structures in an automatic way, which can automatically adjust hyperparameters such as width multipliers and resolution multipliers, reducing a lot of manpower overhead. NAS allows users to balance model size and accuracy according to actual needs (multiple objectives). These models are particularly suitable for resource-constrained environments such as mobile devices and edge computing. For example, lightweight models can more quickly identify and classify product defects in industrial automation, enable fast pedestrian and vehicle recognition in smart cities, and optimize traffic management. In medical devices, they can be used to quickly analyze medical images (such as X-rays and CT scans) to support immediate diagnosis and treatment recommendations.

(2) Model compression. The main technologies of model compression include pruning, quantization, sparse acceleration, etc. They compress original DNNs, enabling them to achieve efficient inference on hardware with limited resources. Pruning aims to reduce redundant model parameters and calculations by removing unimportant weight connections. Quantization technology converts 32-bit floating-point weights into 8-bit or even lower integer representations [39,40] to significantly reduce storage requirements and speed up inference. Sparse acceleration is intended to adapt the pruned sparse model to the hardware. It is closely related to model pruning and sometimes requires joint optimization. In addition, these three methods can be used alone or in combination with each other to achieve the best compression effect. For a mature model with high inference accuracy, its original form may be challenging to deploy directly in fields such as production line monitoring, environmental monitoring, and medical image processing. However, by applying techniques like pruning, quantization, and sparse acceleration, the computational requirements of the model can be significantly reduced while maintaining high accuracy. For instance, we can compress the ResNet101 model to make it suitable for applications in production line monitoring, environmental monitoring systems, or medical image processing equipment, thereby facilitating more efficient practical implementations.

(3) Compilation toolchain. This plays a key role in DNN inference. Representative works include TensorFlow Lite [41], ONNX Runtime [42], and TVM [31], which convert high-level neural network descriptions into efficient execution codes on specific hardware platforms. The compilation toolchain first optimizes the computational graph of the model, such as fusing operations and eliminating redundant nodes. Then, it generates optimized machine code based on the characteristics of the target hardware (such as CPU, GPU, or dedicated accelerator). This process involves the selection of search strategies and hardware instructions, with the goal of maximizing computational efficiency and minimizing latency. An excellent compilation toolchain can not only support models generated by multiple frameworks but also provide consistent performance across platforms. In addition, the hardware and models in actual applications may change continuously according to application requirements (hardware updates, model replacements). Using the compilation

toolchain can quickly achieve the iterative upgrading of applications. For example, compilation tools can be used to quickly deploy and apply various monitoring devices in smart cities and different medical imaging analysis models.

(4) Edge collaborative inference. This uses multiple computing nodes to jointly complete an inference task to improve the overall inference efficiency. Collaborative inference can be achieved through the collaboration of multiple nodes in a distributed system [43,44] or different computing cores in the same device [34,45]. It allows the task to be divided into several sub-tasks and summarizes the results after parallel processing, thereby accelerating the entire inference process. In addition, collaborative inference can also achieve fast preliminary analysis on local devices, and then more in-depth processing by the cloud, ensuring both real-time performance and accuracy. The challenge of edge collaborative inference is how to effectively divide tasks and manage communications between nodes to ensure the balance of task allocation and the efficiency of information interaction. In autonomous driving and smart cockpit applications, models can be deployed on multiple local devices for real-time image processing (such as pedestrian detection). Larger models can also be deployed locally and on edge devices for collaborative processing (such as cockpit voice interaction).

2.3. Structure of the Survey

In order to improve the real-time performance of model inference when edge hardware resources are limited, we must fully consider the balance between model structure characteristics and edge hardware resource characteristics to meet the challenges of the big model era. In summary, the rest of this paper is organized as follows (as shown in Figure 2). In Section 3, we classify experience-based model design and neural network search in lightweight model design, respectively. In Section 4, we summarize three common methods (model pruning, model quantization, and sparse acceleration) in model compression. Then, we classify typical compilation optimization techniques (computational graph optimizations and automatic code generation) in Section 5. In Section 6, we summarize four collaborative inference scenarios related to EI. Finally, we highlight some future research opportunities in Section 7 and conclude the paper in Section 8.

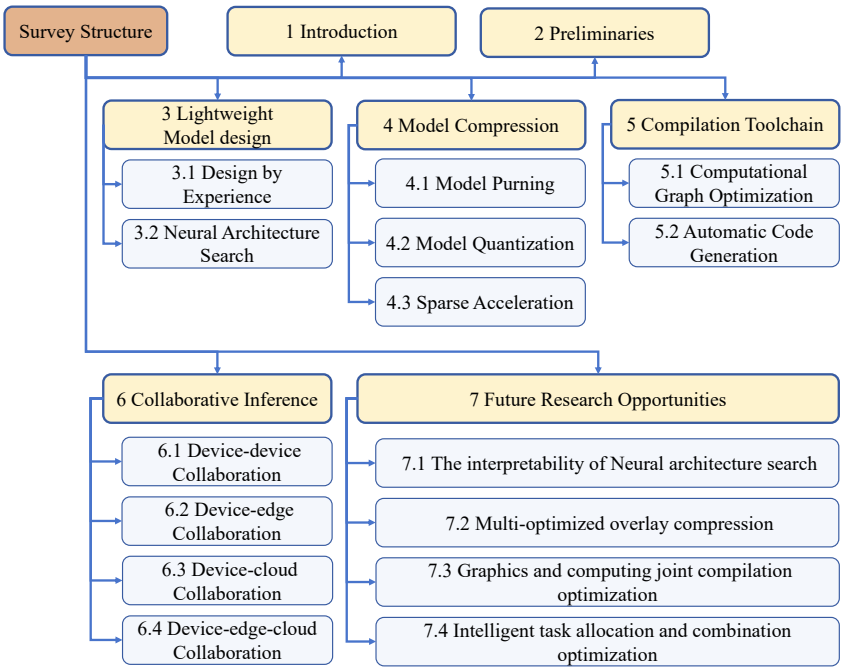


Figure 2. Structure of the Survey.

3. Lightweight Model Design

Lightweight model design aims to develop models suitable for efficient inference on edge devices. During the model construction process, lightweight convolutional modules are meticulously designed to build these models. Compared to traditional convolutional modules, lightweight modules have significantly lower computational requirements and fewer parameters. These optimized models can operate efficiently on mobile devices or IoT devices, helping to conserve device resources and reduce energy consumption. In this section, we categorize lightweight model design approaches into two categories: design by experience and neural architecture search (NAS). We will introduce each method in detail and systematically classify related works.

3.1. Design by Experience

The original lightweight models were designed through human experience, using lightweight convolutions to replace deep convolutions and thereby generate new models [46]. Common lightweight designs include depthwise separable convolution (DSC) and group convolution (GC).

DSC [47] has many significant advantages over traditional standard convolution. First, it decomposes the standard convolution into two steps: depthwise convolution and pointwise convolution. Depthwise convolution applies a filter to each input channel independently for spatial convolution, while pointwise convolution operates on all input channels through 1×1 convolution. This decomposition method greatly reduces the number of parameters and multiplications, thereby reducing computational complexity. Additionally, this reduction in parameters leads to decreased memory usage by the model. Overall, the computational complexity of DSC is $1/n + 1/k^2$ times that of traditional standard convolution.

GC [48,49] offers a series of advantages by dividing both the input and output channels into multiple groups and performing convolution operations independently within these groups. Compared with standard convolution, GC can significantly reduce computational complexity. Because the convolution operations among groups are independent, GC is highly parallelizable. Modern edge hardware can fully utilize this parallelism to accelerate the model inference process. Additionally, different groups in GC can learn distinct types of feature representations, enhancing the model's ability to understand complex data patterns and improve generalization performance. When the input feature channels are divided into G groups in GC, the number of parameters and the computational complexity are each reduced to 1/G of that of standard convolution.

Table 1 summarizes the lightweight models designed for EI. These models have achieved remarkable results in reducing the number of parameters and computational complexity. Specifically, MobileNetV1 [21], MobileNetV2 [47], and EfficientNet [50] utilize specific technologies to reduce the number of required parameters and floating point operations (FLOPs) while ensuring model performance. ShuffleNet [49] uses GC and channel shuffling to effectively reduce the computational burden and improve the efficiency of information flow between different features. AlexNet [37] and CondenseNet [48] also use GC to help reduce the number of parameters and computational complexity. SqueezeNet [22] reduces the amount of computation by compressing and expanding feature maps while maintaining high model accuracy. Overall, these optimization techniques make the model more lightweight and well suited for deployment on mobile devices or other resource-limited environments. In contrast, although VGG19 [51] has fewer model layers (19), it introduces a large number of parameters and high computational cost due to the stacking of standard convolutions.

Table 1. Lightweight models based on experience design.

Optimization Method	Models	Parameters	FLOPS	Top-1 Acc.	Main Technology
Lightweight	1.0 MobileNetV1 [21]	4.2 M	569 M	70.6%	DSC
	1.0 MobileNetV2 [47]	3.4 M	300 M	71.8%	DSC
	EfficientNet-B0 [50]	5.3 M	390 M	77.1%	DSC
	AlexNet [37]	60.9 M	725 M	57.2%	GC
	1.0 ShuffleNetV1 (g = 3) [49]	2.4 M	140 M	68.4%	GC and shuffle
	CondenseNet-86 [48]	0.52 M	65 M	74.9%	GC
	SqueezeNet [22]	1.20 M	837 M	71.1%	Squeeze and expand
Traditional	VGG19 [51]	144 M	19,600 M	72.4%	Standard convolution

Although designing lightweight models based on human experience is important, this approach has several disadvantages, including high time costs, reliance on expert knowledge, difficulty in discovering non-intuitive optimization points, limited generalization ability, and optimization bottlenecks. Specifically, this method frequently requires numerous adjustments and is constrained by the designer's expertise, leading to inadequate adaptability across different tasks and potentially inferior effectiveness compared to automatic search technologies when pursuing maximum efficiency. Therefore, while human design plays a crucial role in the initial understanding and construction of foundational models, it has demonstrated certain limitations within the rapidly evolving field of deep learning.

3.2. Neural Architecture Search

NAS is a search technique that automatically designs optimal neural network architectures using a search algorithm. Automated exploration of optimal neural network architectures not only reduces labor costs but may also uncover novel model structures. NAS can generally be divided into two categories: proxy-based (as shown in Figure 3a) and proxyless (as shown in Figure 3b). Proxy-based NAS relies on proxy models to approximate the performance of the architecture, whereas proxyless NAS performs architecture search and evaluation directly on the target embedded hardware. Both proxy-based and proxyless NAS have their own advantages and disadvantages. We summarize related works on these methods in Table 2.

Table 2. Neural architecture search with multi-metric constraints. We summarize the key indicators for the devices marked with *.

Optimization Method	Model	Target Hardware	Main Technology	Key Metrics	Code Availability
Proxy-based	MnasNet [52]	Google Pixel Phone	A custom weighted product Factorized hierarchical space Reinforcement learning	ImageNet Parameters (3.9 M) Flops (312 M) Accuracy (75.2%) Latency (78 ms)	✓
	Darts [53]	*Mobile setting NVIDIA GTX 1080Ti	Differentiable architecture search Approximate gradient calculation	ImageNet *Parameters (4.7 M) *Flops (<600 M) *Accuracy (73.3%)	✓
	PANS [54]	*Mobile setting GPU (unspecified)	Progressive search Surrogate models	ImageNet *Parameters (5.1 M) *Flops (588 M) *Accuracy (74.2%)	✓
	TreeCell [55]	*Mobile setting GPU (unspecified)	Path-level network transformation Tree-structured architecture Reinforcement learning	ImageNet *Flops (588 M) *Accuracy (74.5%)	✓
	AmoebaNet [56]	NVIDIA Tesla P100	Evolutionary algorithm Hidden state mutation Operation mutation Dentity mutation operations	ImageNet Parameters (86.7 M) Flops (23.1 B) Accuracy (82.8%)	✓

Table 2. *Cont.*

Optimization Method	Model	Target Hardware	Main Technology	Key Metrics	Code Availability
Proxyless	Lyu et al. [57]	NVIDIA Jetson Nano	MobileNetV2-based search space Reinforcement learning	ImageNet Parameters (1.1 M) Accuracy (73.7%) Latency (28.2 ms)	✗
	ProxylessNAS [20]	*Google Pixel Phone NVIDIA TESLA V100 Intel E5-2640 v4	Binarize paths A gradient-based method Reinforce-based algorithm	ImageNet *Accuracy (74.6%) *Latency (78 ms)	✓
	GoldenNAS [58]	*NVIDIA Jetson Xavier NVIDIA Quadro GV100 Intel Xeon Gold 6136	Dynamic channel scaling Progressive space shrinking Evolutionary algorithm Daptive BN Self-knowledge distillation	ImageNet *Accuracy (76.2%) *Latency (52.7 ms)	✗
	PIT [59]	*GAP-8 RISC-V STMicroelectronics-STM32H7	Trainable masking parameters Regularization	PPG Task *Parameters (<5.4 K) *Flops (<293.5 K) *Accuracy (94.13%) *Latency (1.26 ms)	✓
	HGNAS [19]	*NVIDIA Jetson TX2 NVIDIA RTX 3080 Intel i7—8700K Raspberry Pi 3B+	GNN performance predictor A fine-grained hierarchical space A multi-stage hierarchical strategy	ModelNet40 *Parameters (1.48 M) *Accuracy (92.2%) *Latency (36.3 ms)	✗
	Akin et al. [60]	Google Tensor SoC	PPE service NAS integration GC-IBN	ImageNet Accuracy (79%) Latency (26 ms)	✓

(a) Proxy-based NAS (b) Proxy-less NAS

Figure 3. Proxy-based and proxyless NAS.

3.3. Proxy-Based NAS

Proxy-based NAS simplifies the complexity of the original model by introducing a proxy model (as shown in Figure 3a), thereby making the search process more efficient. This approach can significantly reduce the demand for computing resources and accelerate the search process without a significant loss of accuracy. Additionally, this method enables the simultaneous evaluation of multiple candidate architectures, enhancing the efficiency of exploring various architectural spaces.

MnasNet [52] adopts an automated NAS method based on reinforcement learning. It integrates actual inference latency into the objective function and uses a decomposed hierarchical search space to constrain metrics such as accuracy and inference latency. Compared with MobileNetV2, it achieves 1.8 times faster inference speed.

DARTS [53] transforms the architecture selection problem into a differentiable optimization problem by continuously relaxing the architecture search space, using validation set loss as the optimization target. During the search process, constraints are placed on the model's accuracy, perplexity in language modeling tasks, number of parameters, and computational cost. DARTS achieves comparable accuracy to traditional methods while using fewer computational resources.

PANS [54] starts from simple models and gradually explores more complex structures. It uses surrogate models (RNN or MLP) to predict the performance of cell structures. During the search process, PANS constrains metrics such as accuracy, the number of evaluated models, and the number of samples used to train the models. The searched model maintains good classification accuracy while significantly reducing computing resources.

TreeCell [55] adopts path-level network transformation and a bidirectional tree-structured reinforcement learning meta-controller to search for model architectures in a tree-structured space, constrained by metrics like validation set accuracy and the number of model parameters. On the CIFAR-10 dataset, with limited computational resources (about 200 GPU-hours) for training 500 networks, the designed models show better results compared to the original models.

AmoebaNet [56] adopts an improved evolutionary algorithm to explore new model architectures in the search space through hidden state mutation, operation mutation, and identity mutation operations. During the search process, AmoebaNet constrains metrics such as validation set accuracy, computational cost (FLOPs), and the number of parameters.

Although proxy-based NAS can enhance search efficiency, it depends on the accuracy of the proxy model, which might result in suboptimal architectures. Constructing and fine-tuning the proxy model can also introduce additional overhead. Consequently, identifying an appropriate proxy model can be challenging, thereby limiting the applicability of this approach.

3.4. Proxyless NAS

Proxyless NAS abandons the traditional proxy model and searches for neural network architectures directly on the target task and hardware (as shown in Figure 3b), ensuring high accuracy and applicability of the search results. This method brings the model optimization process closer to the actual application scenario, thereby avoiding performance deviations or suboptimal solutions associated with proxy models.

ProxylessNAS [20] transforms the NAS problem into a path-level pruning process. It reduces memory consumption through path binarization. ProxylessNAS trains architecture parameters using a gradient-based method and handles non-differentiable latency metrics by modeling latency or using the REINFORCE algorithm. This effectively demonstrates the effectiveness of ProxylessNAS's direct search approach.

Lyu et al. [57] adapts a multi-objective NAS method based on reinforcement learning to search architectures within the search space, with a design based on MobileNetV2. It guides the search process through a reward function to achieve a balance between efficiency and performance. The model's inference latency on the target edge device (NVIDIA Jetson TX2) is 28.2 ms, significantly better than other comparison models. For example, the inference latency of MobileNetV1 is 54.2 ms, MobileNetV2 is 43.1 ms, NASNet is 78.1 ms, MnasNet is 39.7 ms, and ProxylessNAS-R is 34.6 ms.

GoldenNAS [58] uses dynamic channel scaling, progressive space shrinking, hardware performance modeling, and evolutionary algorithms for architecture search. Additionally, GoldenNAS employs ABN and SKD techniques to enhance the model's adaptability and accuracy. Experimental results show that GoldenNAS has 1.3% higher accuracy than ProxylessNAS and significantly lower search costs.

PIT [59] simultaneously constrains model accuracy (classification accuracy, MAE in regression tasks), the number of model parameters, and the number of inferences. It uses trainable mask parameters to generate binary masks to explore the architecture hyper-parameter space and combines two regularization terms to guide the search direction. Compared with manually designed models, PIT can reduce latency and energy consumption by 5.5 times and 3.8 times, respectively, while ensuring inference accuracy.

HGNAS [19] uses a GNN-based hardware performance predictor, fine-grained hierarchical design space, and multi-stage hierarchical search strategy to search for graph neural architectures on edge devices. In experiments, HGNAS showed significant performance improvements in inference efficiency and reduced peak memory usage. Meanwhile, it effectively balances accuracy and efficiency. Moreover, HGNAS's predictor achieves high

accuracy in predicting hardware efficiency on different devices, and the multi-stage search strategy significantly improves search efficiency.

Akin et al. [60] uses an inverted bottleneck (IBN) variant based on GC to construct the search space, making full use of GC's advantages in adjusting convolution parameters and the number of operations while combining PPE services with diversified NAS algorithms. The model achieved improved accuracy under constraints of multi-dimensional indicators, reducing model inference time and hardware energy consumption by optimizing both model structure and hardware utilization.

However, proxyless NAS requires comprehensive training and evaluation of each candidate architecture, which increases both computational cost and time consumption. Due to the lack of a fast evaluation mechanism, the exploration efficiency of large-scale search spaces is relatively low, making the process of finding the optimal architecture more time consuming. Additionally, proxyless NAS is highly dependent on computing resources, leading to higher economic costs, which may not be practical for small research teams or enterprises. To reduce the computational cost of proxyless NAS, several strategies can be adopted: (1) employ path-level pruning techniques to reduce unnecessary architecture search space and focus on the most promising network structures; (2) utilize binary parameter methods to replace full-precision structural parameters, thereby significantly reducing GPU memory usage and computational requirements; or (3) combine gradient-based methods with reinforcement learning algorithms to further improve search efficiency while ensuring the quality of search results.

3.5. Summary

Empirical design methods and NAS construct lightweight models from two distinct directions. We summarize and analyze these approaches in this section. In summary, empirical design methods feature high interpretability, allowing designers to clearly control the model structure and flexibly customize it according to application requirements. However, they rely heavily on expert knowledge, making the design process time consuming and challenging when exploring complex or innovative structures. NAS offers a high degree of automation, enabling the discovery of novel structures within a vast model space and adapting to various task requirements. Nevertheless, NAS also has notable drawbacks, including high computational resource consumption, stringent hardware requirements, and extended search times. Therefore, both empirical design methods and NAS contribute to enhancing the execution efficiency of models on edge hardware. To enable NAS to rapidly generate efficient models for different hardware devices, designs should focus on creating a flexible search space that includes diverse operators and structures to accommodate varying computing requirements and hardware characteristics. Additionally, incorporating hardware-aware optimization goals such as latency, energy consumption, and resource utilization ensures that the generated models perform well in terms of accuracy while guaranteeing efficient inference. Using cost models instead of direct hardware measurements can further accelerate the NAS search process. This approach enables NAS to effectively produce highly compatible and optimized models for a wide range of hardware platforms.

4. Model Compression

The fundamental difference between model compression and lightweight model design lies in the former's focus on compressing original models using techniques such as parameter pruning, low-bit quantization, and sparsity acceleration. The primary objective is to minimize model parameters and computational complexity while maximizing retained performance metrics (e.g., accuracy). This section provides a systematic review of representative approaches in model pruning, quantization, and sparse acceleration. By categorizing

these methods according to target hardware and technical characteristics, this survey offers a comprehensive overview of the state-of-the-art advancements in model compression.

4.1. Model Purning

The lottery ticket hypothesis [61] posits that there exists a sub-model capable of achieving comparable accuracy without requiring more training iterations than the original model. Neural network pruning involves evaluating model parameters using a targeted algorithm to prune less important parameters while preserving model accuracy, thereby streamlining the neural network's optimization. Consequently, this approach helps mitigate the issue of model over-parameterization [62] and enhances inference efficiency.

In this section, we classify existing neural network pruning methods into three types based on their pruning patterns: structured pruning (as shown in Figure 4a), unstructured pruning (as depicted in Figure 4b), and semi-structured pruning (as illustrated in Figure 4c). Table 3 systematically summarizes the relevant research within each pruning category, providing a detailed overview across four critical dimensions: target hardware platforms, core technical methodologies, constraint metrics (e.g., accuracy, latency), and code availability.

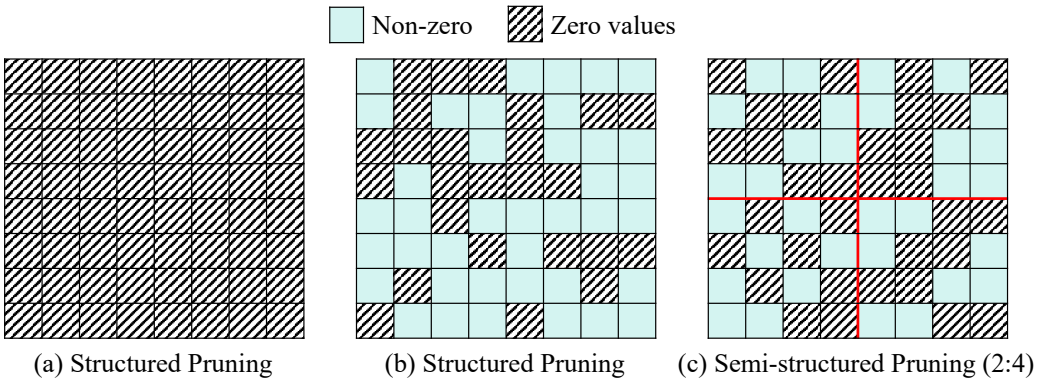


Figure 4. Examples of Structured pruning, unstructured pruning, and semi-structured pruning.

Table 3. Structured pruning, unstructured pruning, and semi-structured pruning.

Optimization Method	Method	Target Hardware	Main Technology	Constraint Indicators	Code Availability
Structured	FlexPruner [63]	NVIDIA Jetson TX2 NVIDIA Jetson Nano	Greedy strategy Iterative perception	Accuracy (−1.12%) Flops (59.8%) Pruning rate (50%)	✗
	AHC [64]	NVIDIA Tesla V100 Intel Xeon Gold 6258R SOPHON BM1604	Unified hardware-aware Multi-objective evolution	Speed up (1.27×) Speed up (1.84×) Parameter (61.1%) Accuracy (−1.0%)	✓
	MyML [65]	Snapdragon 855 Google TPU	Transfer learning Bottom-up	Model size (43%) Speed up (2.93×) Accuracy (−<1%)	✗
	PruneFL [66]	Raspberry Pi 4	Two-stage distribute	Training time (66%) Flops (66%) Accuracy (80%)	✓
	TECO [67]	NVIDIA Jetson TX2 NVIDIA Jetson Xavier	Cross-dimension evaluation Intra-dimension evaluation	MACs (25.6%) Accuracy (73.07%)	✓
	Yang et al. [23]	Snapdragon 888 Kryo 680 Octa-core CPU	Pruning parameterization Soft mask representation	Parameter (−3.3 M) Speed up (1.37×) Accuracy (37.5%)	✗

Table 3. *Cont.*

Optimization Method	Method	Target Hardware	Main Technology	Constraint Indicators	Code Availability
Unstructed	Yu et al. [68]	Avnet Ultra96v2	Hyperparameter introduction Systolic array	Accuracy (−1.4%) Pruning rate (93.75%) Power (−>66%)	✓
	Edge-LLM [69]	NVIDIA Jetson TX2 Meta Quest Pro	Hierarchical unified compression Layer adjustment and voting	Accuracy (+1.29%) Memory (25%)	✓
	u-Ticket [70]	Simulator	Workload balance	Pruning rate (98%) Hardware utilization (2×) Latency (−76.9%) Energy cost (−63.8%)	✓
	Reconvene [71]	Simulator	Initialization pruning	Accuracy (91.26%) Pruning rate (98%)	✗
	ReaLPrune [72]	Fujitsu ReRAM	ReRAM cross-bar array	Training time (5.07%) Accuracy (90.66%) Pruning rate (95.5%)	✗
Semi-structured	CRISP [73]	GPU (unspecified)	Hybrid sparsity Class-aware saliency scores	Accuracy (95%) Pruning rate (90%) Latency (−92.86%) Energy cost (−96.7%)	✓
	Chou et al. [74]	GPU (unspecified)	General pattern set selection Progressive pruning	Accuracy (−0.45%) Pruning rate (54.12%) MACs (55.6%)	✗
	PACA [75]	SIMD PE array	Pattern pruning Channel fusion	Accuracy (−0.87%) Speed up (5.53×)	✗
	All-in-One [76]	Snapdragon Adreno 650	Parametric pruning Switchable thresholds	Accuracy (67%)	✗
	DPACS [77]	XILINX ZCU102	Mask generation	Accuracy (92.15%) MACs (43.8%) Speed up (1.6×)	✓
	KRP [78]	XILINX XC7Z035FFG676-2I	Row-level pruning LR tracking retraining	Accuracy (−0.8%) Pruning rate (66.7%) Resource (−50%)	✗
	SWPU [79]	Customized chip	Hybrid shape and line pattern Dynamic workload balancing	Energy cost (−73.12%) Pruning rate (50.1%) Speed up (4.69×)	✗

4.1.1. Structured Pruning

Structured pruning (as shown in Figure 4a) employs methods such as channel importance assessment [23,67] and layer-level activity monitoring [63,66] to analyze the importance of channels, convolutional kernels, and even entire network layers within the model. Based on predefined rules, it structurally eliminates unimportant model components. For example, if a particular channel in a model layer has minimal impact on overall inference accuracy, it can be removed.

FlexPruner [63] employs a greedy strategy for filter pruning, dynamically adjusting pruning ratios across layers during the process. Through iterative loss-aware pruning and model fine-tuning, it effectively minimizes accuracy loss while meeting compression constraints, achieving efficient compression and acceleration of neural networks on intelligent edge devices. Using the CIFAR-10 dataset as an example, FlexPruner demonstrated a 1.12% decrease in ResNet-32 model accuracy and a 59.8% reduction in FLOPs at the highest pruning rate of 50%. The acceleration ratio on various intelligent edge accelerators ranged from 1.02 to 1.19 times.

AHC [64] leverages unified hardware-aware pruning (UHP) to formulate structured pruning problems across multi-hardware deployment scenarios as multi-objective optimization tasks. By integrating a multi-objective evolutionary solver (MOES), it efficiently generates hardware-efficient models that achieve optimal trade-offs among algorithmic cost, latency, and accuracy. For the ResNet-101 model on the ImageNet dataset, the accuracy decreased by only 1.0% after pruning with the AHC method, while achieving a 1.84× acceleration and a 61.1% reduction in parameters.

MyML [65] leverages transfer learning to investigate both symmetric and asymmetric structured channel pruning techniques, simultaneously performing inference and pruning with shared computation. This approach achieves a significant reduction in model size while accelerating inference speed. It also reduces energy consumption while maintaining accuracy comparable to the original model. On the ImageNet dataset, after pruning, ResNet-50 reduced the model size by 4.3 times on mobile CPUs and achieved an acceleration ratio of 2.93 times, and its accuracy decreased to 73.2% (a decrease of less than 1%).

PruneFL [66] employs a two-stage distributed pruning framework with adaptive pruning strategies to dynamically adjust model size, achieving a substantial reduction in training time while maintaining convergence accuracy comparable to the original model. On the Conv-2 model of the FEMNIST dataset, the time required for PruneFL to achieve 80% accuracy is less than one-third of that required by conventional federated learning (FL), resulting in over 33% time savings. Additionally, FLOPs are reduced by a similar margin.

TECO [67] introduces a multi-dimensional pruning framework that incorporates an inner-dimensional evaluation strategy (INES) to efficiently evaluate the local significance of units within each dimension. Additionally, it employs an inter-dimensional evaluation strategy (ITES) to comprehensively assess these units by considering their global importance. Through subsequent pruning and fine-tuning, this approach achieves optimal trade-offs between model efficiency and accuracy, maintaining high accuracy while reducing runtime latency across diverse embedded platforms. On the ImageNet dataset, compared to the original ResNet50 model, the TECO-S model reduced MAC by approximately 74.4% (from 4.1 billion to 1.05 billion) after pruning, achieving a top-1 accuracy of 73.07%. Compared to GAL-1, the TECO-S model improved top-1 accuracy by 3.25%. On the NVIDIA Jetson TX2 platform, the inference delay was reduced by approximately 50% compared to GAL-1, achieving both higher accuracy and faster inference speed.

Yang et al. [23] proposes a bilevel-optimization-based pruning parametrization method. This method integrates soft masks combined with thresholding and the straight-through estimator (STE) technique to directly perform practical pruning. Under identical computational constraints, it achieves the highest mean intersection-over-union (mIoU) while striking a superior balance between mIoU and inference speed. On the ADE20K dataset, compared to the original model, taking the Ours Small model as an example, and compared to TopFormer Small, when the computational complexity (MACs) is fixed at 1.2 GMACs, the number of parameters is reduced (from 3.1 M to 3.3 M), and the accuracy (mIoU) is improved by 1.4% (37.5% vs. 36.1%). On the Samsung S21 phone, it achieves an inference speed of 75.2 FPS, which is significantly faster than TopFormer Small's 54.7 FPS.

Structured pruning significantly streamlines subsequent storage and computational workflows. By preserving a regularized model structure, it eliminates the need to alter computational or scheduling paradigms during inference, thereby enhancing model deployment efficiency to a certain extent. However, structured pruning methodologies also exhibit notable limitations: (1) Structured pruning units (such as channels, convolutional kernels, or network layers) inevitably remove some valuable local information. Excessive pruning can lead to substantial accuracy degradation. (2) The removal of critical convolutional kernels or channels requires extensive model retraining, thereby increasing training costs.

4.1.2. Unstructured Pruning

Unstructured pruning, as depicted in Figure 4b, focuses on eliminating insignificant individual weights within the model, rather than removing larger structural units such as entire channels, filters, or layers, which is characteristic of structured pruning. This pruning approach allows for a more meticulous reduction of model redundancy. It relies on elaborate sensitivity analysis [71,72], gradient information [68], and statistics-based

criteria [69] to precisely determine the contribution of each connection weight to the model's performance, thereby achieving a higher compression ratio. For example, typical unstructured pruning methods can remove over 90% of the model's parameters with almost no loss in accuracy [80].

As early as 2015, Han et al. [81] proposed a threshold-oriented pruning strategy. Through this strategy, the number of parameters in VGG-16 could be compressed from 138 M to 10.8 M, achieving a significant compression by orders of magnitude. Meanwhile, they established a highly influential three-step pruning paradigm that remains in use to this day. The first step is to train the initial dense network. The second step is to prune the less important connections using specific criteria. The third step is to retrain the network for fine-tuning to restore the model's accuracy.

As early as 2015, Han et al. [81] proposed a threshold-oriented pruning strategy. Using this strategy, the number of parameters in VGG-16 could be compressed from 138 million to 10.8 million, achieving a significant reduction by an order of magnitude. Meanwhile, they established a highly influential three-step pruning paradigm that is still in use today. The steps are: (1) train the initial dense network; (2) prune less important connections using specific criteria; and (3) retrain the network for fine-tuning to restore the model's accuracy.

Yu et al. [68] proposes a hardware-oriented unstructured fine-grained pruning strategy. By introducing hyperparameters N, M, and P, it adjusts the model compression rate. While reducing power consumption, it enhances computational density. Meanwhile, it demonstrates advantages in high-performance, lightweight edge applications. On the ImageNet dataset, when the pruning rate is 16 times, the number of parameters is compressed to 1/16 of the original VGG-16 model (the model size is reduced from 553.44 MB to 34.60 MB), and the top-1 accuracy decreases to 68.7% (a reduction of 1.4 percentage points).

Edge-LLM [69] employs layer-wise unified compression (LUC) technology to minimize computational overhead and reduce memory usage via adaptive layer adjustment and a voting-based pruning scheme. This approach achieves faster inference speeds and improved memory efficiency without compromising accuracy. The Edge-LLM framework is optimized for the LLaMA-7B model, achieving an accuracy improvement of 1.29% compared to the baseline method on the MMLU dataset and similar perplexity to LoRA optimization on the WikiText-2 dataset. Additionally, it reduces memory overhead by a factor of 4 and decreases inference latency by up to 2.92 times.

Based on the lottery ticket hypothesis (LTH), u-Ticket [70] balances the workload by monitoring and adjusting the weight connections of the spiking neural network (SNN) during the training-pruning-initialization process of LTH. With similar accuracy levels, it can boost the hardware utilization rate to 100%, reducing both the operational latency and energy consumption. Compared with the standard LTH method, the u-Ticket method shows a slight decrease in the accuracy of VGG-16 and ResNet-19 networks, while maintaining a pruning rate of approximately 98% and similar parameters. The accuracy of VGG-16 and ResNet-19 networks decreases slightly (such as VGG-16 on CIFAR10 decreasing from 91.0% to 90.7%), while significantly improving hardware utilization to 100%, and significantly reducing operational latency (up to 76.9%) and energy costs (up to 63.8%).

During model initialization, Reconvene [71] performs unstructured pruning to a specified extent. It then assesses the sensitivity of each layer to pruning and prunes the elastic layers. These layers are subsequently re-initialized to ensure optimal performance during subsequent training phases. By doing so, Reconvene achieves a high compression rate while maintaining accuracy, thereby enhancing training and inference speeds on edge devices. Compared to the original model, Reconvene achieves up to 16.21 times compression while maintaining high accuracy (for example, 91.26% for VGG-16, 89.2% for ResNet-20, and only slight changes for ResNet-50) at a pruning rate of 98%.

ReaLPrune [72] integrates the key attributes of the resistive random-access memory (ReRAM) cross-bar array structure and hardware-aware mapping strategy to implement a coarse-to-fine pruning approach. It first prunes at the filter level, then progressively applies channel and index pruning. This method achieves an average pruning rate of 95.5% on redundant CNN weights, reducing hardware requirements and improving training speed on multi-core architectures. Compared to the original model, ReaLPrune achieves a pruning rate of 95.5%, while maintaining high accuracy (for example, ResNet-18 achieves 90.66% accuracy after pruning on the CIFAR-10 dataset). It reduces hardware requirements by 77.2% and achieves a CNN training acceleration ratio of 19.7 times.

Unstructured pruning methods can significantly reduce the number of parameters while maintaining model accuracy. However, these methods also have notable drawbacks: (1) By randomly pruning weights, they introduce irregular parameter distributions, resulting in varying degrees of sparsity. (2) During model deployment, this irregular sparsity prevents most edge hardware from fully utilizing their parallel computing capabilities. Consequently, additional optimization efforts are required to adapt the sparse model to target platforms, thereby increasing deployment complexity.

4.1.3. Semi-Structured Pruning

Semi-structured pruning, also referred to as pattern pruning, addresses the limitations of both structured and unstructured pruning by striking a balance between them. Building on structured pruning frameworks, this approach performs fine-grained parameter optimization while satisfying predefined constraints. Unlike unstructured pruning, which removes individual weights, semi-structured pruning targets weight parameters that follow a specific structural pattern. For example, in 2:4 pruning [82], two non-zero values are retained for every four adjacent convolutional kernel parameters (as shown in Figure 4c). By leveraging analytical techniques such as module-activity-based analysis [77,78] and feature map redundancy analysis [73,74], semi-structured pruning identifies and systematically removes redundant structural modules within weight matrices. This methodology effectively balances the compression ratio and computational efficiency while maintaining acceptable performance levels.

CRISP [73] employs a gradient-based class-aware saliency score to guide an iterative pruning strategy, achieving a hybrid-structure pruning that combines fine-grained N:M sparse patterns with coarse-grained block-sparse patterns. While maintaining comparable accuracy, it achieves a high compression ratio and reduces both latency and energy consumption. When the pruning rate of the CRISP method reaches 90% on the VGG-16 model, the accuracy still exceeds 95%. Compared to other methods, it can reduce latency by up to 14 times and energy consumption by up to 30 times.

Chou et al. [74] proposes a method for selecting a set of general patterns, choosing 16 high-frequency and symmetric pattern sets. It adopts a two-stage pruning strategy, considering the pruning of the first layer separately. By designing a progressive pruning framework, it effectively balances the computational load, model size, and accuracy loss in the hardware deployment of CNN models. On the ImageNet dataset, using PHO to prune CNN models, the 3×3 convolution model achieved a compression rate of 2.18× and reduced computational complexity by 2.25 times. For example, after pruning HarDNet-68, the top-1 accuracy loss was only 0.45%.

PACA [75] proposes a flexible pattern pruning algorithm. It compresses the number of patterns in convolutional neural networks using the alternating direction method of multipliers (ADMM). By leveraging pattern pruning and channel fusion techniques, it reduces the idle rate of processing elements (PEs). Consequently, while reducing the overhead of index storage, it enhances hardware performance. PACA achieves a hardware

performance improvement of 2.01 to 5.53 times with a small precision loss (for example, a top-1 accuracy loss of 0.87% for VGG16 on CIFAR-10).

All-in-One [76] proposes a combination of parametric pruning, switchable threshold, and batch normalization methods. With extremely low memory consumption, it achieves high accuracy across models with varying sparsities. Additionally, it significantly reduces the variance in inference latency caused by changes in execution frequency, addressing the issues of limited hardware resources and unstable energy support in edge devices. At the same time, the All-in-One framework maintains high accuracy even at high pruning rates (for example, ResNet-32 achieves over 94% accuracy on CIFAR-10 and ResNet-18 exceeds 67% accuracy on ImageNet).

DPACS [77] dynamically generates spatial and channel pruning masks using input activations. These masks are shared across layers within a block, thereby reducing the computational overhead associated with mask generation. This approach not only significantly accelerates end-to-end network inference but also maintains high accuracy levels. After pruning, ResNet-Cifar achieves an accuracy of 92.15% on CIFAR-10, while ResNet-101 achieves a 1.78 times reduction in MACC and a 1.60 times acceleration ratio at s25-c25.

Based on the row scale of convolutional kernels, KRP [78] integrates a retraining method based on learning rate (LR) tracking. This approach achieves a balance between a high pruning rate and high accuracy across various CNN models. It significantly reduces storage requirements and hardware resource consumption on FPGAs, while also improving computational parallelism. KRP achieves an accuracy loss of less than 0.8% at a pruning rate of 66.7%. When combined with GSNQ quantization, it can compress network model storage by 27 times, reducing on-chip hardware resource consumption by more than half on FPGAs.

SWPU [79] introduces a hardware–software co-design framework that employs a hybrid sub-structured pattern pruning (HSPP) algorithm. This approach iteratively prunes models using a combination of shape and line patterns, enabling high sparsity while maintaining hardware compatibility. By optimizing both architectural and computational efficiency, it enhances peak energy efficiency and reduces energy consumption, thereby facilitating the deployment of DNN training on power-constrained edge devices. Compared to structured pruning, SWPU achieves a 50.1% higher pruning rate, reaching a maximum frequency of 675 MHz and peak energy efficiency of 126.04 TFLOPS/W on an FPGA. When training ResNet-18 models, it saves 3.72 times the energy and accelerates by 4.69 times compared to previous sparse training processors.

Structured pruning, unstructured pruning, and semi-structured pruning each have their own advantages and disadvantages. Unstructured pruning allows for highly refined weight deletion, achieving a high compression rate, but its irregular sparse pattern makes it difficult to effectively utilize hardware accelerators, resulting in low computational efficiency. Structured pruning simplifies the model by removing entire channels, layers, or other structural units in the neural network. While the compression rate is lower, it is easier to accelerate on existing hardware, although this may lead to performance degradation. Semi-structured pruning aims to combine the benefits of both methods by pruning in blocks or small matrices, offering a compromise solution, but it involves more complexity in optimizing the pruning granularity. Empirical studies have shown that the effectiveness of these methods varies across different application scenarios. Selecting the most suitable method requires considering multiple factors, including model accuracy, computing resources, and deployment platform characteristics. Additionally, a comparative study [80] among the three pruning methods indicates that no single method performs best in all cases; thus, the choice should be guided by specific project requirements and constraints.

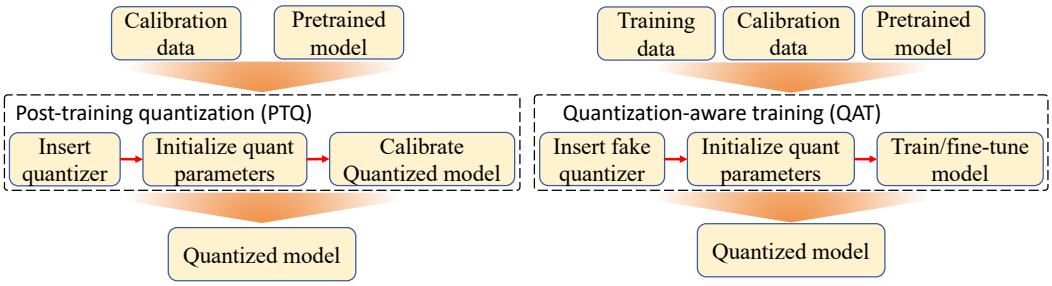
4.2. Model Quantization

In the training and deployment of traditional neural networks, model parameters are typically stored using 32-bit or 16-bit floating-point numbers. This imposes significant storage and computational demands on resource-constrained devices like mobile terminals and embedded systems. Model quantization leverages the observation that low-precision data representations can adequately preserve the computational behavior of high-precision models. Specifically, by converting 32-bit floating-point weights and activations to 8-bit integers, this technique maintains model accuracy while significantly reducing the memory footprint. Additionally, it takes advantage of the computational efficiency of integer arithmetic to accelerate inference, making it especially suitable for deployment on resource-constrained edge devices.

In this paper, we categorize existing neural network quantization approaches into two distinct categories based on the quantization phase: post-training quantization (PTQ) (illustrated in Figure 5a) and quantization-aware training (QAT) (as depicted in Figure 5b). Table 4 provides a systematic summary of representative studies under each quantization paradigm, offering a comprehensive overview across five critical dimensions: target hardware platforms, core technical methodologies, constraint metrics (e.g., accuracy, computational cost), quantization bit-width, and code accessibility.

Table 4. Post-training quantization and quantization aware training.

Optimization Method	Method Name	Target Hardware	Main Technology	Constraints	Quantization Bit-Width	Code Availability
PTQ	Liu et al. [83]	Mobile setting	Ranking loss Nuclear norm	Accuracy (81.29%) Memory (75%)	4–10	✓
	PTQ4DM [84]	GPU (unspecified)	NDTC calibration method MSE quantization metric	IS (+15.52) FID (−24.92) sFID (−17.36)	8	✓
	EasyQuant [40]	Rockchip RK3399	Scale optimization ARM NEON ISA	Accuracy (68.26%) Computational cost (−33%)	8, 7, <7	✗
	AFP [85]	GPU (unspecified)	Adaptive floating-point format Bayesian optimization	Accuracy (−0.04%) MACs (10.75%) Energy cost (92.42%)	3.9–5	✓
	ZeroQuant [39]	GPU (unspecified)	Kernel fusion Layer-by-layer knowledge distillation	Speed up (2.6×) Memory (33%)	8, 4/8	✓
	AWQ [86]	NVIDIA Jetson-Orin Nano NVIDIA RTX 4070	Activation-aware weight protection On-the-fly dequantization Kernel fusion	Accuracy (−<0.1%) Memory (25%) Speed up (3.3×)	4, 3, 16	✓
	Agile-Quant [87]	Snapdragon 870 Raspberry Pi 4B	Activation quantization strategy TRIP matrix multiplication	PPL (6.09) Speed up (2.55×)	4, 8	✗
QAT	LLM-QAT [88]	GPU (unspecified)	Data-free distillation KV cache quantization	Accuracy (69.9%) Model size (25.9%)	4, 6, 8	✗
	Octo [89]	Huawei Atlas 200DK NVIDIA Jetson Xavier	Loss-aware compensation Parameterized range clipping	Accuracy (98.8%) Speed up (2.03×) Peak memory (29.67%)	8	✓
	DAQ [90]	GPU (unspecified)	Distance-aware soft rounding (DASR) Temperature controller	Accuracy (91.2%)	1, 2, 3, 4, 32	✓
	QUANTISENC [91]	AMD Virtex Ultrascale	Variable quantization Dynamic configuration	Accuracy (96.5%)	1.3, 5.3	✓
	MPQ-YOLO [92]	NVIDIA RTX 3090	Trainable scale Progressive strategy	Accuracy (74.7%) Model size (7.04%)	1, 4	✗
	QuantNAS [93]	Kirin 9000	Batch statistics Scale predictor	Accuracy (94.5%) Model size (30%) Latency (−40%)	8	✗
	Lin et al. [94]	STMicroelectronics-STM32F746	Quantization-aware scaling	Accuracy (+1.68%)	8	✗



(a) Post-training quantization (PTQ) (b) Quantization-aware training (QAT)

Figure 5. Quantization-aware training and post-training quantization.

4.2.1. Post-Training Quantization

PTQ is a quantization operation performed after the model training is completed (as shown in Figure 5a). Its main idea is to directly quantize the pre-trained floating-point model, converting the weights and activation values in the model from higher-precision data types to lower-precision data types without the need to retrain the model. This method is simple to implement and fast, enabling the quick conversion of a floating-point model into a quantized model for deployment. It is suitable for scenarios where the impact on model accuracy is relatively minor.

Liu et al. [83] proposed a post-training quantization (PTQ) algorithm designed specifically for vision transformers. This approach incorporates techniques such as ranking loss and nuclear-norm-based mixed-precision quantization. The effectiveness of this method was validated across multiple benchmark models and datasets. Compared to traditional PTQ algorithms, their method not only enhances model accuracy but also reduces memory storage and computational costs. On the ImageNet dataset, using the 8-bit quantization of the DeiT-B model as an example, this approach saves approximately 25% of memory compared to the original model, while achieving a top-1 accuracy of 81.29%.

PTQ4DM [84] provides a comprehensive investigation into the selection of quantization operations, calibration datasets, and calibration metrics specifically tailored for diffusion models. It introduces two key methodologies: the normally distributed time-step calibration (NDTC) method and the mean squared error (MSE) quantization metric. By employing these advanced techniques, PTQ4DM successfully quantizes pre-trained diffusion models to 8-bit representations without compromising their performance. This quantization process not only accelerates the denoising process but also significantly enhances the deployment efficiency of these models. Taking the ImageNet 64×64 image generation task as an example, when using denoising diffusion implicit models (DDIMs) with 100 steps, the model quantized using the PTQ4DM method achieves a FID of 24.92 and an IS of 15.52.

EasyQuant [40] introduces a quantization method that jointly optimizes the scale factors of weights and activations, thereby constructing an INT7 quantization inference framework. This method is designed to be independent of specific hardware, making it versatile for various applications. It significantly improves the accuracy of quantized models across a range of computer vision tasks. Experimental results show that EasyQuant effectively reduces computational costs and inference latency while maintaining high accuracy. These improvements make it particularly suitable for deploying models on embedded hardware platforms such as ARM. The EasyQuant method reduces computational costs by approximately 20% to 33% compared to the original model when quantized to INT7. It exhibits lower inference latency on the RK3399 platform and maintains high accuracy across various tasks and model architectures. For example, MobileNetV1 achieved an INT7 quantization accuracy of 68.26% on the ImageNet 2012 dataset.

AFP [85] introduces a novel quantization format with flexible exponent and mantissa bit-widths, integrating an automatic optimization framework based on Bayesian optimization. This approach achieves minimal accuracy degradation in low-bit quantization while significantly reducing computational and energy costs. Consequently, it demonstrates comprehensive advantages in improving overall model efficiency. When using AFP's 4.8-bit quantization, ResNet-50 experiences a precision loss of 0.04%, and MobileNet-v2 experiences a precision loss of 0.6%. Additionally, the AFP encoding model reduces MAC operations by 9.3 times, while the quantizer saves an average of 13.2 times in computational resources.

ZeroQuant [39] employs fine-grained, hardware-friendly quantization schemes, a low-cost layer-by-layer knowledge distillation algorithm, and optimized quantization Transformer kernels. These techniques enable efficient inference of large-scale Transformer models under low-bit quantization. By reducing the memory footprint and computational costs, ZeroQuant effectively controls accuracy loss and significantly boosts model inference speed. By using INT4/INT8 mixed-precision quantization, the model's memory usage is reduced by three times. On the GPT NeoX 20A model, the GPU requirement can be reduced from two to one, and the inference latency can be reduced from 65 ms to 25 ms. This results in an overall efficiency improvement of 5.2 times.

AWQ [86] preserves salient weights based on activation distributions and mitigates quantization errors through channel-wise scaling. It also incorporates techniques such as on-the-fly dequantization, weight packing, and kernel fusion within the TinyChat framework. This integrated approach enables efficient compression and acceleration of large language models (LLMs) across diverse edge devices. By maintaining model accuracy while significantly reducing memory footprint and enhancing inference speed, AWQ makes the deployment of LLMs on resource-constrained edge platforms both feasible and practical. AWQ performs 4-bit quantization on LLMs, achieving a compression rate of 4 times. Compared to Huggingface FP16, it achieves a speed improvement of 3.2 to 3.3 times on desktop and mobile GPUs.

Agile-Quant [87] introduces three key innovations: an activation-guided quantization strategy, an activation-aware token pruning technique, and an efficient two-refine improved by pruning (TRIP) matrix multiplication framework. This integrated approach enables efficient inference of large language models (LLMs) on edge devices by simultaneously quantizing model weights and activations while preserving task performance. The LLaMA-7B model has a quantified PPL of 6.09 on the Wikitext-2 dataset. Compared to the FP16 model, it achieves up to 2.55 times acceleration on multiple edge devices.

Although the PTQ (post-training quantization) process can map high-precision model parameters and activation values to low-bit representations, thereby reducing the parameter count and computational burden, the non-retraining approach inevitably incurs information loss. This results in a decrease in model accuracy. This accuracy degradation is especially pronounced in ultra-low-bit quantization. When employed for complex tasks and large-scale models, the decline in accuracy may render the model's performance inadequate for practical needs.

4.2.2. Quantization-Aware Training

QAT, also known as quantization during training, integrates the quantization operation into the training process (as depicted in Figure 5b). During training, it simulates the computational process after quantization, enabling the model to adapt to low-precision representations at the training stage. Consequently, the model can better preserve its performance after quantization. In comparison with PTQ, QAT typically attains superior

model performance at the same quantization precision. This characteristic makes QAT well suited for scenarios where high model accuracy is required.

LLM-QAT [88] leverages data-free distillation and quantization-aware training techniques to significantly enhance the performance of large language models under low-bit quantization. By integrating these methodologies, it effectively reduces the memory footprint while maintaining high computational efficiency. This enables viable solutions for efficient LLM deployment on resource-constrained platforms. On the LLaMA-30B model, when quantized to 4 bits using the LLM-QAT method, the model size was compressed from 60.6 GB to 15.7 GB, and the zero-shot average accuracy reached 69.9% under the 4-8-4 setting.

Octo [89] employs loss-aware compensation (LAC) and parameterized range clipping (PRC) methodologies to achieve model quality comparable to full-precision training under 8-bit quantization. By dynamically compensating for quantization errors and adaptively clipping parameter ranges, this approach significantly reduces computational and memory footprints while accelerating both training and on-device inference. On the CIFAR-10 dataset, GoogLeNet trained by Octo achieved a model accuracy of 97.6% to 98.8%, accelerated image processing speed by up to 2.03 times, and reduced peak memory usage by up to 3.37 times.

DAQ [90] utilizes distance-aware soft rounding and a temperature controller to alleviate gradient mismatch and quantizer gap problems within a unified framework. This approach achieves significant improvements in model accuracy compared to other methods across various network architectures and quantization bit-widths, providing a more effective solution for network quantization. When the DAQ method is used for W1A32 quantization of models such as ResNet and MobileNet-V2, the accuracy reaches up to 91.2%, with only a 0.2% decrease compared to the full-precision model.

QUANTISENC [91] utilizes variable quantization and mixed-precision techniques, along with dynamic configuration of neuron parameters. It achieves improved resource utilization, reduced power consumption, and optimized performance per watt on multiple datasets. Meanwhile, it maintains high accuracy across different quantization bit-widths, providing a superior solution for neuromorphic hardware design. The model accuracy of this method can reach 96.5% after quantization on the Spiking MNIST dataset, demonstrating its effectiveness in maintaining high performance even under quantized conditions.

MPQ-YOLO [92] integrates 1-bit backbone and 4-bit head quantization with a trainable scale and the progressive network quantization (PNQ) strategy. By doing so, it strikes an optimal balance between model compression and detection performance, providing an effective approach for deploying real-time object detection models on edge devices. Compared to the original model, the MPQ-YOLO framework achieves a compression rate of up to 16.3 times in computational complexity and up to 14.2 times in model size. Despite these significant reductions, it maintains high detection accuracy, achieving 74.7% mAP@0.5 on the VOC dataset and 51.5% mAP@0.5 on the COCO dataset. This demonstrates its effectiveness in balancing model efficiency with detection performance.

QuantNAS [93] employs a batch-statistics-based training strategy and scale predictor technology. When applied to the Kirin 9000 mobile CPU, it effectively enhances the performance and deployment efficiency of quantized models under various latency constraints. This approach ensures that models can be efficiently deployed on mobile devices while maintaining high performance across different latency requirements. The quantization network searched by QuantNAS demonstrated a top-1 accuracy improvement of 1.53% to 1.68% on the ImageNet 1K dataset and a 1.7% mAP improvement on the COCO dataset.

Lin et al. [94] introduces an integrated algorithm–system co-design framework. Initially, it applies quantization-aware scaling to stabilize 8-bit quantized training, ensuring

numerical stability during low-precision computations. This is followed by sparse update mechanisms that dynamically prune redundant parameters to minimize the memory footprint. Finally, the Tiny Training Engine optimizes computational graphs through kernel fusion and operation scheduling. Through this systematic integration, the framework enables efficient on-device training of IoT devices under a 256 KB memory constraint. It achieves a balance between maintaining model accuracy and reducing computational overhead, while also enabling continuous learning from new data streams. When the model compression rate reaches up to 70%, the accuracy of different models after quantization is maintained between 91.8% and 94.5%, while the inference latency is reduced by 40%.

The main disadvantages of QAT are its high implementation complexity and resource requirements. Since QAT needs to simulate the quantization process during training, this increases the complexity of the training pipeline and may prolong the training time. Additionally, QAT requires modifications to the model, such as adding quantization nodes and careful tuning of hyperparameters. These adjustments place higher demands on the user's technical expertise. Moreover, although QAT can mitigate accuracy loss caused by quantization, performance degradation may still occur in certain scenarios, particularly when the dataset is small or the model is highly complex. Therefore, while QAT can provide better quantization effects, its implementation cost and challenges are relatively high. Users should weigh these factors against the potential benefits when deciding whether to adopt QAT for their models.

4.3. Customized Sparse Acceleration and General Sparse Acceleration

Sparse acceleration enables neural networks to skip zero or near-zero elements in the data, focusing instead on key non-zero elements for efficient computation. This approach significantly reduces unnecessary computational operations, cutting down on data transmission costs and speeding up the network inference process.

In this paper, we systematically categorize existing sparse neural network acceleration methods into two categories based on their level of generality: customized sparse acceleration (illustrated in Figure 6a) and general sparse acceleration (shown in Figure 6b). Table 5 provides a comprehensive overview across four critical dimensions: target hardware platforms, core technical methodologies, performance metrics (e.g., latency, energy efficiency), and code accessibility.

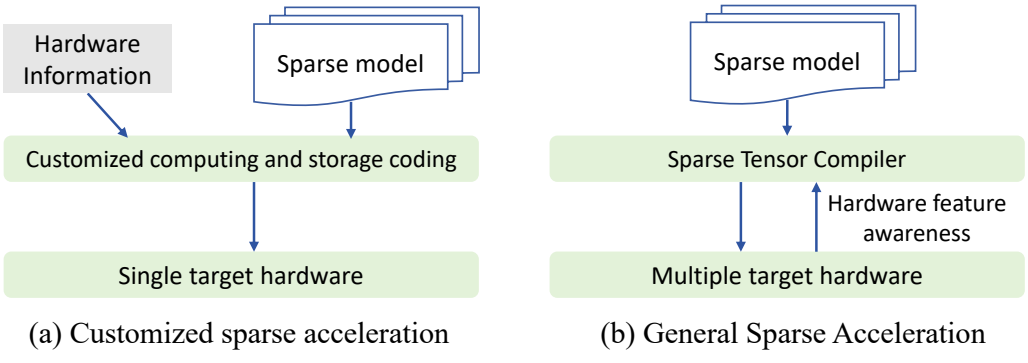


Figure 6. Customized and general sparse acceleration design.

Table 5. Customized sparse acceleration and general sparse acceleration.

Optimization Method	Method Name	Hardware Desgin	Main Technology	Constrained Metrics	Code Availability
Customized	SCNN [95]	Customized chip	PT-IS-CP-sparse dataflow	Speed up (2.7×) Energy cost (43.48%)	✗
	RAMAN [96]	Efinix Ti60	Sparse processing Reconfigurable dataflow	Throughput (13.5 GOP/s) Energy cost (136.96 mW) Peak memory (−37%)	✗
	SNAP [97]	Customized chip	Channel-first dataflow Two-level psum reduction	Speed up (2.87×) Energy efficiency (3.61 TOPS/W)	✗
	SparTen [98]	Terasic DE2-150	Bitmask representation Greedy balance	Speed up (4.3×) Memory (76.92%)	✗
	FixyFPGA [99]	Intel Stratix-10 GX 10M	Fixed-weight design Fully-pipelined activation buffering	Speed up (2.34×)	✓
	STA [38]	Intel Arria 10 SX660 SoC	Diverse matrix multiplication engine (DMME) Scalable softmax module	Energy efficiency (12.28×) MAC efficiency (51×)	✗
	Sparse-T [100]	Ibex RISC-V	Dual-version ASIC design Metadata processing optimization	Speed up (2.1×) Energy cost (47.3%) Occupied area (30.86%)	✗
	Cambricon-X [101]	Customized chip	PE-based architecture Indexing module (IM) Asynchronous compute	Speed up (7.23×) Troughput (544 GOP/s) Energy efficiency (6.34×) Energy cost (954 mW)	✗
	Cambricon-S [102]	Customized chip	Entropy encoding Shared indexing	Speed up (1.71×) Energy efficiency (1.37×) Energy cost (798.55 mW)	✗
General	Taco [103]	Intel Xeon E5-2680 v3 NVIDIA RTX 2080 Ti	Data structure abstraction Sparse iteration space theory	Performance (14×) Correctness	✓
	SparTA [104]	NVIDIA RTX 2080 Ti AMD Radeon VII Intel Xeon Silver 4210	Tensor-with-sparsity-attribute abstraction Sparsity attribute propagation	Inference latency (8.4×) Memory footprint Model accuracy	✓
	SparseTIR [105]	GPU (unspecified) CPU (unspecified)	Composable formats Composable transformations	Speedup (1.52×) Memory footprint	✓
	Tian et al. [106]	Intel Xeon Gold 6126	Unified tensor storage format representation	Performance (6.26×) Code quality	✓
	DynaSpa [107]	NVIDIA Jetson Orin NVIDIA Jetson Xavier Qualcomm Adreno 650	Relaxed sparsity composition Polyalgorithm kernel composition	Performance (4.4×) Search time Runtime cost	✗
	Zhang et al. [108]	Intel Xeon E5-2640v4	Insert-sort-merge template Automatic workspace insert	Performance (27.12×) Memory usage	✗
	Flash-LLM [109]	GPU (unspecified)	Load-as-sparse and compute-as-dense Software pipeline design	Performance (2.9×) Throughout (3.8×)	✓
	SAM [27]	Intel Xeon Silver	Core data model and dataflow blocks Custard compiler	Generality performance Hardware modeling ability	✓

4.3.1. Customized Sparse Acceleration

Customized sparse acceleration pertains to hardware or software optimizations meticulously tailored to specific hardware architectures or application scenarios. This approach aims to maximize the computational efficiency of sparse neural networks. It typically includes the design of specialized sparse storage coding schemes, the implementation of hardware-efficient customized computations, and the formulation of dynamic workload-balancing strategies. By doing so, it can effectively leverage the unique features of specific hardware, such as single instruction, multiple data (SIMD) units [99], or custom-designed chips [95,97,101]. Simultaneously, it seeks to minimize memory access costs and computational overhead.

SCNN [95] implements a PT-IS-CP sparse dataflow architecture. This architecture integrates compressed storage and computation mechanisms to effectively leverage the inherent sparsity in both weights and activation maps of CNNs. By strategically transmitting and processing only non-zero data elements, this approach significantly minimizes redundant computations and reduces data movement overhead through optimized data reuse strategies. Experimental results demonstrate that SCNN achieves superior performance and energy efficiency compared to conventional dense CNN accelerators. Specifically, it delivers a 2.7 times speedup in inference throughput and a 2.3 times improvement in energy efficiency when evaluated under comparable hardware configurations. Compared to the baseline intensive CNN accelerator (DCNN), SCNN achieved performance improvements

of 2.37×, 2.19×, and 3.52× on AlexNet, GoogLeNet, and VGGNet, respectively (with an average of 2.7×). It also reduced energy consumption by 2.3×.

RAMAN [96] maximizes computational efficiency by exploiting sparsity in both activation maps and weights, incorporating a hardware-aware weight pruning strategy. This approach effectively reduces memory storage and access requirements, thereby enhancing computational energy efficiency. Moreover, RAMAN utilizes a reconfigurable dataflow mechanism that selects the optimal data processing method for different layers, significantly reducing data movement costs and computational latency. As a result, RAMAN can support a variety of DNNs. In edge-computing scenarios where low power consumption and resource constraints are critical, it effectively improves inference performance. RAMAN achieved an effective throughput of 13.5 GOP/s and 10.5 GOP/s on the MobileNetV1 and DS-CNN models, respectively, with energy consumption of 136.96 mW and 131.77 mW, and energy efficiency of 98.47 GOP/s/W and 79.68 GOP/s/W. When processing the MobileNetV1 model, the peak activated memory was reduced by 37%, and for the DS-CNN model, it was reduced by 49%. The accuracy on the VWW and KWS tasks was 80.7% and 93.7%, respectively.

SNAP [97] employs a channel-first dataflow architecture to address frontend challenges through parallel associative index matching (AIM) units and sequence decoders. This design achieves an average computational utilization of 75%. For backend optimization, a two-stage partial sum (psum) reduction mechanism is implemented, which reduces psum writeback traffic by 22 times while supporting configurable operations for diverse layer types. A 16 nm test chip implementing the SNAP architecture was evaluated across various workloads. Experimental results demonstrate competitive performance and energy efficiency metrics, validating the effectiveness of the proposed sparse acceleration approach. SNAP has an average acceleration ratio of 2.87 times for sparse ResNet-50 models, with the highest energy efficiency reaching 3.61 TOPS/W.

SparTen [98] efficiently processes sparse data using bit mask representation, avoiding some of the overhead of traditional methods. It utilizes a greedy balancing algorithm to alleviate load imbalance between computing units. This significantly improves SparTen's performance compared to dense and one-sided sparse architectures, while reducing computational and memory energy consumption. Thus, SparTen provides an efficient solution for sparse matrix computation. Compared to dense architecture and single-sided sparse architecture, SparTen has an acceleration ratio of 4.3 times and 1.9 times, respectively.

FixyFPGA [99] hard encodes the weights of the trained CNN network into hardware. It adopts a fully parallel and fully pipelined approach for convolution processing, supporting high-sparsity and low-precision computations. To optimize the model, it employs batch normalization fusion and high-sparsity element-wise pruning. Additionally, the Deep Freeze tool is utilized to automatically generate hardware code. Experimental results indicate that FixyFPGA outperforms previous works in image classification and object detection tasks across multiple datasets. Compared to previous work, FixyFPGA achieves a 2.34-fold higher GOPS in the ImageNet classification task and a 3.82-fold increase in frame rate in the Pascal VOC object detection task.

STA [38] employs the diverse matrix multiplication engine (DMME) to effectively handle various matrix multiplication types, mitigating computational redundancy. By incorporating a scalable softmax module, it eliminates off-chip communication of intermediate results, addressing memory bottlenecks inherent in Transformer architectures. As a result, STA performs outstandingly in terms of latency, energy efficiency, and MAC (multiply-accumulate) efficiency, showing significant advantages over CPUs, GPUs, and other FPGA accelerators. Compared to previous FPGA accelerators, STA has increased energy efficiency by 12.28 times and MAC efficiency by 51.00 times.

Sparse-T [100] uses dual-version ASIC accelerators and optimizes metadata processing to accelerate sparse matrix-dense vector multiplication for low-power microcontrollers. By handling metadata in hardware, it overlaps indexing operations with core computations, reducing the burden on the main CPU core. This approach improves computing speed, achieving a speedup of 1.3 to 2.1 times at different sparsity levels, and results in significant energy savings, ranging from 15.8% to 52.7%. Sparse-T provides an effective hardware acceleration solution for sparse data processing on low-power devices. Additionally, the Sparse-T1 and Sparse-T2 designs occupy 30.86% and 40.09% of the processor area.

Cambricon-X [101] conducts local computations by storing compressed synapses in processing elements (PEs) and uses an indexing module to precisely select neurons, reducing the requirement for data transmission bandwidth. Additionally, it leverages an asynchronous computing mode to enhance overall efficiency. As a result, Cambricon-X performs outstandingly in terms of performance and energy efficiency. Compared to traditional accelerators such as CPUs, GPUs, and DianNao, it achieves an average speedup of 7 times when processing sparse neural networks. Compared to DianNao, Cambricon-X achieves an average acceleration ratio of 7.23 times, reduces energy consumption by 6.43 times, and has a power consumption of 954 mW. The highest achievable speed is 544 GOP/s.

Cambricon-S [102] employs coarse-grained pruning through local convergence observation to mitigate the irregularity of sparse synapses. It further compresses data using entropy coding and adopts shared indexing to reduce storage overhead. Through the synergistic effect of these methods, Cambricon-S effectively enhances the performance and energy efficiency of the accelerator in processing sparse neural networks. Compared to the most advanced sparse neural network accelerator, Cambricon-S achieves a performance improvement of 1.71 times and an increase in energy efficiency of 1.37 times, with a power consumption of 798.55 mW.

However, customized sparse acceleration has certain limitations. First, it is highly dependent on hardware and requires optimization solutions tailored for specific architectures (such as FPGAs and dedicated chips), making it difficult to adapt to a variety of edge devices or new computing platforms. Second, it needs continuous updates to keep pace with hardware iterations and the evolution of software frameworks, necessitating significant long-term investment.

4.3.2. General Sparse Acceleration

General sparse acceleration (as shown in Figure 6b) typically employs sparse tensor compilers to achieve sparse acceleration across multiple hardware platforms. The sparse tensor compiler is a tensor-level compiler specifically designed for sparse irregular operators in deep learning. Its goal is to provide a general programming abstraction that covers various deep learning workloads, including graph neural networks, sparse transformers, sparse convolutions, and network pruning. Additionally, it leverages hardware-aware optimizations to generate high-performance code tailored for different hardware platforms.

Taco [103] adopts methods such as data structure abstraction, sparse iteration space theory, tensor notations, and a compilation process to build a powerful tensor algebra compiler. It can compile various sparse tensor algebra expressions into code with performance comparable to hand-optimized code on CPUs and GPUs. Taco provides developers with an efficient and general-purpose tool, significantly improving the efficiency and flexibility of sparse tensor computations. The Taco tool is 14–1600 times faster in performance than MATLAB Tensor Toolbox.

SparTA [104] introduces the tensor-with-sparsity-attribute (TeSA) abstraction, extending traditional tensor abstraction to describe sparse attributes and patterns. SparTA infers

the sparse attributes of other tensors in the model through attribute propagation and performs execution plan conversion, generating efficient code based on this information. SparTA can significantly reduce inference latency and memory consumption. It also accelerates sparse model training and promotes sparse model exploration, providing a unified and scalable framework for studying sparsity in deep learning models. Compared to seven baselines, such as PyTorch (v1.7) JIT and TensorRT (v7.2), SparTA accelerates inference latency by an average of 1.7 to 8.4 times across different DNN models and sparse modes.

SparseTIR [105] provides a sparse tensor compilation abstraction with composable formats and transformations, optimizing performance by building a search space that includes these components. It features three IR stages and designs related compilation processes. In multiple experimental environments, SparseTIR outperforms function libraries in both single-operator and end-to-end scenarios. However, it requires further improvements in automatic scheduling, format decomposition, dynamic sparsity processing, and integration with graph-level IR. Compared with cuSPARSE, dgSPARSE, etc., SparsetIR improves speed by 1.20–2.34 times on GNN operators, 1.05–2.98 times on sparse attention operators, 0.56–7.45 times on sparse convolution operators, 1.08–1.52 times on end-to-end GraphSAGE training, and 4.20–40.18 times on RGCN inference.

Tian et al. [106] proposes a domain-specific language (DSL) and a compiler. The DSL uses high-level programming abstractions similar to Einstein notation to represent tensor algebra operations and supports multiple tensor storage formats. The compiler introduces a new sparse tensor algebra (TA) based on MLIR, employing input-related code optimizations to improve data locality. During the compilation process, the DSL is first converted into TA, which is then gradually transformed into machine code. Experimental results show that the kernels generated by this compiler outperform those produced by the TACO compiler. Compared with TACO, in sequentially executed SpMM and SpMV, the acceleration ratio is as high as 6.26× and 2.14×, while in parallel-executed SpMV, the acceleration ratio is as high as 20.92×.

DynaSpa [107] introduces Dense In Sparse Tile (DISTile) to optimize memory access patterns and data reuse. It utilizes a multi-algorithm kernel automatic scheduler to search for candidate kernels offline and selects the optimal kernel at runtime, reducing search space with the help of performance analysis models. DynaSpa introduces acceptable runtime overhead without compromising model accuracy. However, there is still room for improvement in the accuracy of device performance prediction models and support for other types of sparsity. Compared to traditional libraries and tensor compilers, the acceleration ratios on Jetson AGX Orin GPU, Jetson AGX Xavier GPU, and Adreno mobile GPU for handling DNN operators with 50–90% spatial sparsity are 1.3–4.4 times, 1.6–7.7 times, and 1.5–7.8 times.

Zhang et al. [108] addresses the sparse scattering problem by introducing sparse workspaces as an efficient adapter between computational code and result tensors. The approach proposes an insertion sort merge (ISM) algorithm template, which serves as the core of the code generation algorithm and features modular characteristics. The compiler can automatically detect sparse scattering behavior in tensor expressions and insert necessary intermediate workspace tensors. Overall, the code generated by this compiler achieves performance and memory usage efficiency comparable to hand-optimized libraries when processing sparse tensor algebraic expressions. Compared to the previous intensive workspace, the speed has increased by up to 27.12 times.

Flash-LLM [109] is the first efficient software framework that supports exploring unstructured sparsity on high-performance tensor cores. It adopts a "sparse loading, dense computing" approach to improve matrix multiplication efficiency by reducing memory access bottlenecks and tolerating redundant calculations. Additionally, Flash-LLM uses the

Tiled-CSL sparse format, sparse-to-dense conversion, and a two-level overlapping strategy to perform sparse data extraction and computational overlap. Experiments show that Flash-LLM's core-level performance far exceeds that of Sputnik and SparTA. At the kernel level of SpMM, the average acceleration ratio is 2.9 times and 1.5 times higher than that of Sputnik and SparTA. At the end-to-end framework level of the OPT-30B/66B/175B model, compared to DeepSpeed and FasterTransformer, the number of tokens generated per GPU per second has increased by up to 3.8 times and 3.6 times.

SAM [27] supports both reconfigurable and fixed-function spatial dataflow accelerators. It defines spatial dataflow graphs capable of expressing all computations of sparse tensor algebra. SAM also designs dataflow primitives tailored to the fundamental features of sparse tensor algebra, representing multi-dimensional sparse and dense tensors as flat streams with hierarchical control tokens. Furthermore, the paper proposes a compilation strategy that translates high-level tensor index representations into the SAM model. This approach enables efficient execution on spatial dataflow architectures.

To adapt optimization methods to different hardware platforms, a general sparse accelerator should focus on model sparsity processing and employ a unified hardware adaptation strategy. It needs to include a set of intermediate representations (IRs) that can express sparse structures and allow optimization conversions for various hardware platforms. Additionally, it should utilize a hardware abstraction layer (HAL) to uniformly manage the characteristics of different hardware platforms, such as computing unit types, memory hierarchies, and data transmission mechanisms, enabling the compiler to dynamically adjust optimization strategies. Furthermore, integrating an automated tuning mechanism ensures optimal performance and efficiency. By addressing these aspects, the general sparse accelerator can effectively handle sparse models and provide flexible, efficient support across diverse hardware platforms.

4.4. Summary

This chapter delves into three mainstream model compression methods, aiming to reduce model parameters and computational complexity while preserving performance metrics. Model pruning can effectively cut down redundant parameters and computations. However, structured pruning may remove crucial local information, unstructured pruning can introduce irregular parameter distributions, and semi-structured pruning still faces challenges in balancing accuracy and computational efficiency in practical applications. Quantization techniques can reduce memory footprint and computational burden significantly. Nevertheless, PTQ inevitably leads to information loss, and quantization-aware training, although offering better performance, has a more complex training process. Sparse acceleration can decrease unnecessary computational operations. Customized sparse acceleration is highly hardware-dependent, while general sparse acceleration requires further enhancement in multiple aspects. In the context of the rise of large models, model compression techniques remain a hot research topic.

5. Compilation Toolchain

In this section, we will discuss EI acceleration methods from the perspective of compilation optimization, including computational graph optimization and automatic code generation. Both of them aim to improve the execution efficiency of DNNs while reducing the consumption of computing resources, so that DNNs can run more efficiently on different hardware platforms.

5.1. Computational Graph Optimization

The computational graph is a representation used by existing deep learning frameworks to model DNNs. It consists of various types of operators. Each operator has different dimensional attributes (such as dimensionality information and memory layout). The core of graph optimization involves fusing and splitting operators to eliminate redundant computations and improve memory access efficiency (as illustrated in Figure 7). Current graph optimization techniques can be categorized into two main groups. The first group includes frameworks like TensorFlow and TensorRT, which optimize the computational graph based on predefined rules and strategies developed by engineers, such as fusing convolution and activation functions, pointwise convolution with activation functions, and simplifying mathematical expressions equivalently. These methods utilize equivalent transformations during graph optimization to improve performance while maintaining mathematical accuracy. Although fully equivalent transformations can significantly enhance inference performance, they are constrained by the limited scope of the search space.

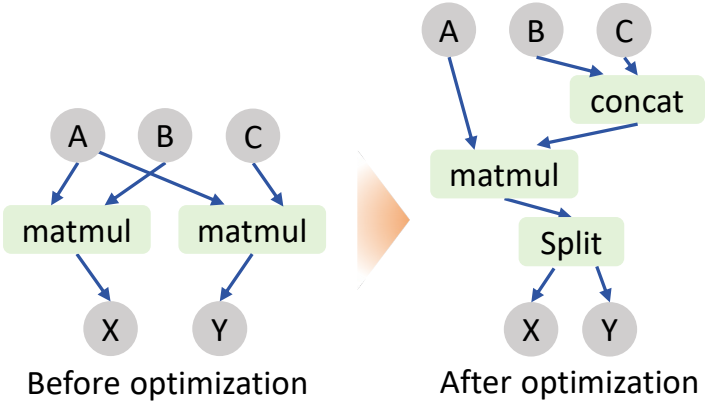


Figure 7. A example of subgraph optimization.

The second category comprises approaches that employ automated optimization of computational graphs, such as TASO, PET, and MetaFlow, which use search strategies to identify deeper opportunities for fusion and splitting within a given search space. During this process, both equivalent and non-equivalent transformations may be applied. For non-equivalent transformations, all identified subgraphs are validated against rules based on mathematical equivalence. Only those subgraphs that show performance improvements while preserving mathematical equivalence are selected for substitution. Automated graph optimization provides a significant advantage over rule-based strategies by effectively broadening the search space, thus uncovering more substantial opportunities for graph optimization at deeper levels. Table 6 summarizes the current mainstream efforts in automated graph optimization. While some of these works focus primarily on optimizing computational graphs for general-purpose platforms, they can be easily adapted to edge hardware with automated code generation capabilities.

TVM [31] employs operator fusion and data layout transformation techniques for graph optimization. It identifies various types of graph operators and formulates fusion rules to combine multiple small operations into a single operation, thereby reducing memory access and computational overhead. Additionally, it specifies optimal operator layouts based on the hardware memory hierarchy and performs transformations when necessary to enhance data access efficiency.

Table 6. Computational graph optimization.

Model	Target Hardware	Main Technology	Key Metrics	Code Availability
TVM [31]	ARM Cortex A53 XILINX Artix-7 ARM Mali-T860MP4 NVIDIA Titan X	Tensor expression language Operator fusion Data layout transformation	Inference time Resource utilization	✓
MetaFlow [110]	NVIDIA Tesla V100 NVIDIA Tesla P100	Relaxed graph substitutions Multi-dim. cost model Graph split algorithm	Inference time Resource utilization	✓
TASO [30]	NVIDIA Tesla V100	Graph substitutions Formal verification Joint optimization	Inference time	✓
PET [29]	NVIDIA Tesla V100	Data layouts transformation Partial equivalence transformation Automated corrections	Inference time Resource utilization	✓
EINNET [111]	NVIDIA Tesla V100 NVIDIA Tesla A100	Tensor algebra expression Derivation rules	Inference time	✓
DNNFusion [112]	Intel Xeon E5-2680 v4 Samsung Galaxy S20 Samsung Galaxy S10 Honor Magic 2	Fusion opportunity analysis Mathematical-based graph rewriting Profile-driven fusion plan	Inference time Resource utilization Compilation time	✓
Chimera [113]	Intel Xeon Gold 6240 NVIDIA Tesla A100 Huawei Ascend 910	Block decomposition and reordering Intra-block optimization	Inference time Cache utilization	✓
Welder [114]	NVIDIA Tesla V100 NVIDIA RTX 3090 AMD MI50 GPU Graphcore IPU	Tile graph construction Hierarchical scheduling Code generation and optimization	Inference time Memory access	✓

MetaFlow [110] employs techniques such as relaxed graph replacement, cost-based backtracking search algorithms, and flow-based recursive graph segmentation. It relaxes the performance constraints of graph replacements and explores a larger computational graph space. During this search process, it utilizes a multi-dimensional cost model to assess the performance of the computational graph and partitions large graphs into optimizable subgraphs to enhance inference performance. Compared with existing deep learning frameworks like TensorFlow, TensorFlow XLA, and TensorRT, MetaFlow achieves inference speed improvements ranging from 1.1 to 1.6 times across various DNN models.

TASO [30] jointly optimizes graph replacements and data layouts using an extended backtracking search algorithm and a cost model. It performs graph replacements by enumerating computational graphs and utilizing hash tables, and verifies their correctness using first-order logic and an automatic theorem prover. Experiments demonstrate that TASO enhances inference performance and reduces kernel launch overhead. TASO can accelerate inference by 1.3 to 2.8 times compared to the cuDNN backend and by 1.1 to 1.8 times compared to TVM.

PET [29] performs graph optimization using partial equivalent transformations and automatic correction. The search process mutates operator types based on a depth-first search algorithm. The automatic correction process employs a simplified equivalence check using first-order logic. It utilizes box propagation, random testing, and the generation of correction kernels to ensure program functional equivalence. The program optimizer splits the program, identifies the optimal solution by combining complete and partial equivalent transformations, and subsequently applies post-optimization operations such as inverse elimination, operator fusion, and preprocessing. Compared with existing frameworks like TensorFlow, TensorFlow XLA, TensorRT, and TASO, PET can increase inference speed by up to 2.5 times.

EINNET [111] employs an inference-based optimization method to represent tensor programs as tensor algebraic expressions. It transforms these expressions using intra-expression and inter-expression inference rules. During the instantiation of expressions,

it applies operator-matching techniques to align suitable expressions with predefined operators. The remaining expressions are then passed to a pre-existing kernel generator. Finally, it utilizes a distance-guided search algorithm combined with fingerprint technology for redundancy pruning. Compared with the best existing baseline framework, EINNET can enhance inference performance by 1.2 to 2.72 times.

The primary technologies of DNNFusion [112] include fusion opportunity analysis, mathematical property graph rewriting, lightweight contour-driven fusion search, and fusion code generation and optimization. These optimization techniques enable DNNs to perform efficient inference on mobile devices, discover additional operator fusion opportunities, and reduce compilation time. On mobile CPUs, DNNFusion can increase inference speed by 1.4 to 2.6 times compared to TASO across different models.

Chimera [113] employs block decomposition and reordering for graph optimization. It breaks down computationally intensive operators into computational blocks and utilizes analytical models to determine the optimal execution order. Additionally, it takes into account multi-level memory hierarchies to minimize data movement costs. During internal block optimization, replaceable microkernels are used to abstract computational blocks, and specific microkernel code is generated for different hardware backends. Compared with PyTorch + CuDNN, Relay + TensorRT, Relay + CuDNN, and Relay + Ansor, Chimera achieves an average inference speed improvement of 1.42 times, 1.31 times, and 1.22 times, respectively.

Welder [114] optimizes deep learning memory access by introducing tile abstraction. The optimization includes defining computations base on tile-level tasks and building tile graphs to manage dataflow. A hierarchical scheduling strategy is adopted to decompose the optimization space and independently optimize tiles at different memory layers. Hardware-aligned tile searches are performed taking into account hardware-related factors. WELDER can significantly discover new fusion patterns, and optimize computing efficiency. Compared with PyTorch, ONNXRuntime, Rammer, Ansor, TensorRT, and FasterTransformer, WELDER can achieve a minimum acceleration of 1.11 times and a maximum acceleration of 4.29 times.

5.2. Automatic Code Generation

Computation graph optimization simplifies the computation graph structure while enhancing computational efficiency (as shown in Figure 8). The optimized computation graph features clearer logic and a more regular structure. Automatic code generation tools can more conveniently process the optimized computation graph and map it to specific hardware platforms. For instance, the code generator can more easily identify which parts can be executed in parallel. Additionally, the performance results obtained after code generation can validate the effectiveness of the computation graph optimization and provide feedback for further improvements to the optimization algorithm. Table 7 summarizes the current mainstream automatic code generation works, which effectively accelerate the final executable code for various hardware platforms based on their unique characteristics.

Wang and Shen [115] primarily focus on TensorCore and Simba in deep learning accelerators and conduct automatic kernel generation for large language models. They build a high-quality action space by formulating the kernel generation problem as a reinforcement learning task. The approach employs a transformer policy network and the REINFORCE algorithm to enable the agent to autonomously learn kernel parameters. Additionally, it utilizes variance reduction techniques (such as batch sampling and hyperparameter tuning) to enhance the stability and performance of the algorithm. Wang and Shen [115] achieve an average speedup of 3.5 times on TensorCore compared to exploration-based Ansor and 2.6 times on Simba compared to solver-based CoSA.

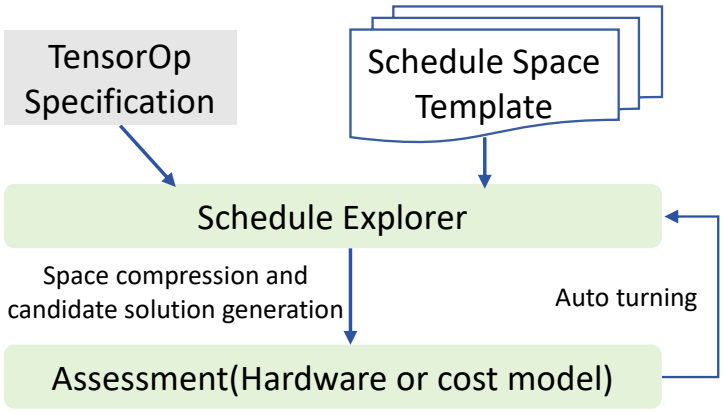


Figure 8. Automatic code generation optimization.

Table 7. Automatic code generation.

Model	Target Hardware	Main Technology	Key Metrics	Open Source
Wang and Shen [115]	GPU (unspecified)	Reinforcement learning Variance reduction	Code performance Energy efficiency	✓
FastConv [116]	Huawei Kunpeng 920 Snapdragon 835, 855, 888 Apple M1 Amazon Graviton2	Winograd algorithm Tensor transformation C++ automatic generation	Code performance Cache utilization	✓
Danopoulos et al. [117]	XILINX Alveo U200	Heterogeneous streaming Parallel processing	Code performance	✗
Fu and Huang [118]	Intel Xeon (R) Gold 6126	ACG programming model Dataflow graph IR Code template	Code performance Memory consumption	✗
Zhao et al. [119]	FT-Matrix DSP	Loop transformation Instruction-level optimization Reinforcement learning	Code performance	✗
Welder [114]	NVIDIA Tesla V100 NVIDIA RTX 3090 AMD MI50 GPU Graphcore IPU	Tile-graph abstraction Two-step scheduling algorithm Hardware mapping	Code performance Memory access	✓
Ansor [120]	Intel 18-core 8124M NVIDIA Tesla V100 Raspberry Pi 3b+	Hierarchical search space Program sampling Cost model	Code performance	✓
Rammer [121]	NVIDIA Tesla V100 AMD Radeon Instinct MI50 Graphcore IPU	Gradient descent algorithm rOperator abstraction vDevice abstraction rTask-aware DFG compiler	Code performance Hardware utilization Scheduling overhead	✓
AKG [32]	Huawei Ascend 910	Polyhedral transformations, Tiling and fusion strategies Vectorization Low-level synchronization	Code performance	✓
DIETCODE [122]	NVIDIA Tesla T4	Shape universal search space Cost model Joint learning Local filling optimization	Code performance Scheduling overhead	✓

FastConv [116] primarily focuses on the automatic code generation of high-performance convolution kernels for ARM CPUs. It optimizes the computational efficiency of tensor multiplication using an improved Winograd algorithm and the TensorGEMM library. Template-based automatic generation and automatic tuning techniques are employed to produce optimized code tailored to different hardware models. Compared with NNPACK, ARM NN, FeatherCNN, and TVM, FastConv can enhance inference performance by 1.02 to 28.36 times.

The work of Danopoulos et al. [117] is based on the HLS4ML project and proposes a general optimization scheme for FPGAs, specifically targeting the XILINX Alveo U200. It

automatically converts ONNX models into FPGA kernel code and performs optimizations at the kernel level, memory level, and host level.

Fu and Huang [118] employ the ACG programming model for multi-core CPU systems, enabling users to easily define GNN models via an init function and three UDFs. It constructs a dataflow graph intermediate representation (IR) to represent the GNN model and incorporates sub-tensors using a deferred execution mechanism. During the code generation phase, it generates C++ kernel function code based on the IR and code templates, applying various memory management strategies for intermediate tensors.

Zhao et al. [119] implement automatic code generation through JOKer, which integrates loop transformations, vectorization, and instruction-level optimizations into an end-to-end optimization process. It employs a reinforcement learning algorithm based on Q-learning to enable the agent to continuously explore from a random initial configuration. Meanwhile, it updates the Q-table based on the execution cycle reduction values returned by the hardware.

Welder [114] performs kernel generation based on TVM, utilizing parallel compilation and subgraph caching to enhance compilation speed. It incorporates hardware optimization information during the code generation and compilation processes. For instance, on NVIDIA GPUs, Welder considers memory access merging and compute core utilization when determining efficient data tile sizes. Welder achieves more than an order of magnitude faster code generation compared to Ansor.

Ansor [120] defines a hierarchical search space (consisting of sketches and annotations) and generates programs through random sampling. It then employs evolutionary search and learned cost models to iteratively fine-tune these programs. Additionally, Ansor utilizes a gradient-descent-based scheduling algorithm to allocate time resources. On various hardware platforms, it significantly enhances the execution performance and search efficiency of deep learning tasks compared to existing methods. Compared with the PyTorch, Halide, FlexTensor, and AutoTVM frameworks, Ansor improves the performance of generated inference code by up to 3.8 times, 2.6 times, and 1.7 times, respectively, and is more than an order of magnitude faster than AutoTVM.

Rammer [121] abstracts operators into rOperators (which are composed of multiple rTasks to expose intra-operator parallelism) and hardware accelerators into vDevices (each containing multiple vEUs and incorporating barrier-rTasks to ensure execution order). Additionally, Rammer employs an rTask-aware DFG compiler (providing scheduling interfaces, analyzers, and a wavefront-scheduling-based strategy) to achieve automatic code generation for deep learning computations. Compared with TensorFlow, TVM, TensorFlow-XLA, and TensorRT, Rammer improves performance by 2.18 to 33.94 times on NVIDIA GPUs and by 5.36 to 41.14 times on AMD GPUs.

AKG [32] converts TVM's DSL expressions into polyhedral IR. It uses a polyhedral scheduler to solve the ILP problem for loop transformations and optimizations. During tiling, it adopts a reverse strategy to construct the tile shape. The tile sizes are automatically determined based on hardware specifications. Depending on data unloading and forking conditions, it applies different fusion strategies. AKG utilizes the scheduling tree to achieve automatic storage management. It transforms convolution operations into a fractal GEMM kernel to optimize computations. In the code generation stage, it performs vectorization and low-level synchronization optimizations. Additionally, it employs a machine-learning-guided sampling-based automatic tuning strategy. This strategy enables automatic code generation for the Huawei Ascend 910 chip. AKG is primarily compared with manually optimized code and TVM. It achieves an average speedup of 1.6 times over TVM on individual operators, and an average speedup of 1.3 times and 5.6 times on subgraphs

compared with TVM and manually optimized code, respectively. The overall performance of the end-to-end network is improved by 20.2% compared to TVM.

DIETCODE [122] designs a general search space and a microkernel-based cost model. This model decomposes the cost of the complete program into shape-independent and shape-dependent components. Additionally, DIETCODE employs a joint learning optimization workflow, enabling all workload instances to share space and cost models. Meanwhile, DIETCODE implements local filling optimization, which addresses bounds-checking issues during data reading and writing. The performance of the code automatically generated by DIETCODE is up to 69.5% higher than that of Ansor and up to 18.6% higher than that of NVIDIA's related libraries. The automatic scheduling time is reduced by up to 94.1 times compared to Ansor.

5.3. Summary

This section introduces two compilation acceleration technologies: computation graph optimization and automatic code generation. We summarize some of their recent work. In general, the goals of computation graph optimization and automatic code generation are both to enhance the execution efficiency of deep learning models. In practical deep learning compilers, these two processes typically work closely together. Computation graph optimization generates efficient computation graphs for automatic code generation, while automatic code generation transforms the optimized computation graphs into executable code. They interact with each other to ensure that the optimization strategies can be tailored to different hardware platforms.

6. Collaborative Inference

Collaborative inference refers to the process of executing complex inference tasks through the cooperation of multiple devices, enhancing both the accuracy and efficiency of DNN inference (as shown in Figure 9). This approach leverages the distinct advantages of various devices while compensating for the limitations inherent to each device, thereby achieving more efficient inference in complex scenarios. In this section, with a focus on device-centric collaboration, we categorize collaborative inference into four types: device–device collaboration, device–edge collaboration, device–cloud collaboration, and device–edge–cloud collaboration. We will subsequently introduce and summarize pertinent research efforts in each of these areas from recent years.

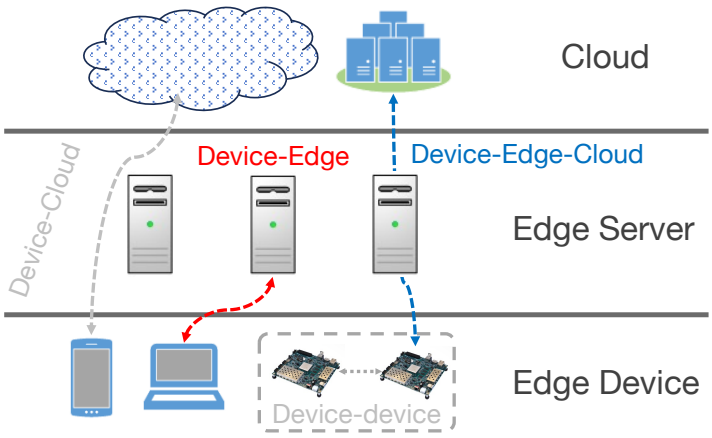


Figure 9. An illustration of the collaborative inference.

6.1. Device–Device Collaboration

Device–device collaborative inference embodies a cooperative framework among multiple resource-limited end devices, such as sensors and smartphones, operating inde-

pendently of centralized cloud servers. This model offers several key advantages: it ensures low latency, facilitates data localization, and promotes decentralization. Additionally, it imposes minimal demands on communication bandwidth, supports operation in offline environments, and enhances privacy protection.

Within the context of device–device collaboration, prevalent technologies mainly include model partitioning [34,45,123], task assignment [14,34,124–126], and ensemble-based methodologies [127,128], among other approaches. Table 8 offers a general summary of pertinent works, listing the supported models, hardware platforms, core technologies employed, key performance indicators, and availability of source code.

Table 8. Device–device collaboration.

Work	Supported Models	Platforms	Main Technology	Key Metrics	Code Availability
DeViT [13]	ViT DeiT CCT	NVIDIA Jetson Nano	Knowledge distillation	Accuracy Latency Energy Power	✗
RFS [123]	CNN	NVIDIA RTX 2080Ti NVIDIA GTX 1080Ti NVIDIA Jetson Xavier	Model partitioning	Accuracy Latency Service reliability	✓
Galaxy [124]	DistilBert Bert-L GPT2-L OPT-L OPT-XL	NVIDIA Jetson Nano	Hybrid model parallelism Communication optimization	Latency Scalability	✗
HALP [14]	VGG-16	NVIDIA GTX 1080Ti NVIDIA Jetson Xavier	Task partitioning	Speedup ratio Throughput Service reliability	✓
HALP (extended) [125]	MobileNet-v1 VGG-16	Raspberry Pi 4	Task partitioning	Latency Accuracy Service reliability	✗
COIN-LEO [34]	Self-built DNN	Simulation	Model partitioning Task assignment PPO	Throughput Latency Network overhead	✗
Edge ensembles [127]	MobileNet-v2	Unmentioned	Communication optimization Ensemble aggregation Vector quantization	Latency Accuracy	✓
[126]	VGG-16 ResNet-34	Raspberry Pi 3B+ Raspberry Pi 4B NVIDIA Jetson TX2	Workload assignment	Execution time	✗
[128]	ResNet-18	Unmentioned	Model aggregation	Accuracy Latency	✗
[45]	AlexNet VGG-19 YOLONet	Unmentioned	Model partitioning	Throughput Execution time	✗

Xu et al. [13] propose DeViT, a collaborative inference framework for ViTs in edge devices. The large ViT is decomposed into multiple small models to reduce inference latency and energy consumption while maintaining comparable accuracy. They design a decomposition-and-ensemble algorithm named DEKD based on knowledge distillation, which fuses multiple small models, minimizes accuracy loss, and reduces communication overheads. A feature-matching module is also developed to promote the learning of decomposed models from the large ViT. This approach overcomes the limitations by not just focusing on reducing parameters and FLOPs but also truly optimizing for latency and energy. The DeViT framework can, on average, reduce inference latency by 1.90×, cut energy consumption by 26.11%, and only sacrifice 2% accuracy compared to large ViTs, while outperforming lightweight ViTs in terms of latency, energy consumption, and accuracy trade-off.

Li et al. [123] propose methods to accelerate distributed CNN inference in collaborative edge computing. To avoid accuracy loss in task partitioning, they introduce receptive-field-based segmentation (RFS), which partitions the input tensor considering the receptive field, stride, and padding of each convolutional layer, ensuring the same inference accuracy as the pre-trained model. To reduce computation time and communication overhead, a

novel collaborative edge computing with fused-layer parallelization is presented. It divides the CNN model into multiple fused blocks. After computing each block, only a small fraction of sub-outputs needs to be exchanged among edge servers. Moreover, dynamic programming (DPFP) is used to find the optimal partitioning solution, which can balance communication and computation time effectively. The proposed methods can significantly improve inference acceleration by up to 73% compared to the pre-trained model, reduce communication overhead, and ensure high service reliability of over 99.999% under time-variant channels while maintaining the same inference accuracy as the pre-trained model.

Ye et al. [124] introduce Galaxy to address the resource-intensive nature of Transformer inference on edge devices. Galaxy's workflow involves three main stages: pre-processing, parallel planning, and execution. During pre-processing, calibration data are used to record runtime information on edge devices. In parallel planning, a hybrid model parallelism (HMP) architecture is adopted, combining tensor parallelism and sequence parallelism to reduce data dependencies and communication overhead. Moreover, heterogeneous and memory-aware workload planning is carried out to minimize latency and prevent out-of-memory errors. In addition, tile-based communication optimization is applied to overlap communication and computation, reducing synchronization delays. Galaxy achieves up to 2.5× end-to-end latency reduction and demonstrates excellent scalability, outperforming state-of-the-art approaches in various edge environments.

Li et al. [14] focus on accelerating distributed CNN inference in edge computing. To ensure inference accuracy during task partitioning, the receptive field is considered when using segment-based partitioning. A novel task collaboration scheme named HALP is designed. Before computing the first convolutional layer, the host ES partitions the input image. Two parts of the partitioned data are sent to secondary ESs, and the host ES processes the overlapping zone. During the computation, the host ES and secondary ESs work in a coordinated manner. For each convolutional layer, while the host ES transmits data, it computes the overlapping zone output and sends it to secondary ESs. Secondary ESs calculate the output needed by the host ES and their own required output simultaneously. HALP is also extended to the multi-task scenario, enabling the host ES to manage tasks from multiple IoT devices more effectively. The HALP method can accelerate CNN inference of VGG-16 by 1.7×–2.0× for a single task and 1.7×–1.8× for four tasks per batch on specific devices and enhance service reliability under time-variant channels compared to the state-of-the-art MoDNN.

Dong et al. [125] focus on the extended design of the distributed CNN inference acceleration method called HALP in edge computing. It aims to address the issue of CNN inference in IoT devices with limited processing capabilities by partitioning the input tensor of each convolutional layer along the largest dimension. To avoid accuracy loss caused by stride and padding in traditional methods, it adopts RFS. Through seamless collaboration among edge devices, it maximizes the parallelization between communication and computation. The HALP method implemented with three Raspberry Pi 4 devices achieves a 1.7× inference acceleration for VGG-16, and by combining with CNN model compression and dynamic model selection, it can further optimize the trade-off between latency and accuracy, significantly improving service reliability in time-critical IoT applications compared to stand-alone computation.

Xu et al. [34] focus on the collaborative inference problem in low-Earth-orbit (LEO) satellite edge computing for unmanned aerial vehicles. The proposed COIN-LEO algorithm aims to efficiently split models and assign tasks among satellites. Firstly, it selects appropriate satellites to participate in collaborative inference, which reduces the sum of inter-satellite link propagation delays and adapts well to network changes. Secondly, it uses deep reinforcement learning (DRL) with the proximal policy optimization (PPO) algorithm

to split DNNs. A neural network is designed to predict submodel inference times based on satellite resource occupation, which enables more accurate model splitting and task assignment, optimizing inference performance. Compared with single-node inference, the COIN-LEO strategy improves the inference throughput by 85.80%, and compared to the equally splitting algorithm, it shows an average improvement of 17.285% in inference time, highlighting its superiority in enhancing inference efficiency.

Malka et al. [127] present the "edge ensembles" mechanism, aiming at the limited computational resources of edge devices and the drawbacks of existing methods. Each device uses a local model with a shared encoder and unique decoder, enabling diverse model sets for independent and collaborative tasks. During inference, the device first encodes and quantizes the input data and broadcasts them to neighboring devices. Neighboring devices process the data and then return the results. The initiating device aggregates these results to obtain the prediction. To reduce communication overhead and latency, they adopt the VQ-VAE model to quantize features and propose two collaborative inference methods. A weighted aggregation scheme is also used to balance the influence of the local model and the decisions of neighboring devices, thus enhancing the inference accuracy. Experiments show that the edge ensembles method can improve the accuracy from about 75% of a single device to over 80% in some cases, like the CIFAR-10 dataset, with a minor additional delay compared to local inference.

Chen et al. [126] present EdgeCI, a low-latency distributed inference framework for CNNs in edge clusters. It uses two main methods, the auction-based workload assignment scheme and the fused-layer parallelization strategy based on non-recursive dynamic programming. The former treats workload assignment as an auction between extended workload partitions and IoT devices, which ensures a more balanced workload distribution; the latter determines the optimal set of fused blocks for CNN model partitioning with low complexity and computation–communication balance. Both two methods are plug-and-play, improving the efficiency on edge clusters without additional training or deployment. EdgeCI can improve the inference speed by 34.72–43.52% compared to typical state-of-the-art solutions.

In the work [128] on enhancing edge ensembles, Kumazawa et al. investigate three conventional model integration techniques: cascade, weighted averaging, and test-time augmentation (TTA). To mitigate the cascade's increased latency while reducing computational complexity, they propose the m-parallel cascade, facilitating parallel processing of models to decrease latency and adjust computational load. Concerning weighted averaging and TTA, ensemble label-based learning is introduced, utilizing ensemble prediction labels, which is particularly advantageous in edge environments where labeled data are scarce. They also propose an accuracy-based weighting method to optimize computational resource use, improving the overall performance of the collaborative inference system. The proposed m-parallel cascade reduces latency by 2.8 times compared to the conventional cascade with a 1.06-fold increase in computational costs, and weighted averaging and TTA improve accuracy in specific scenarios, such as up to 0.7% in CIFAR-100 for single-model TTA and up to 0.8% in HEE setting with ACC-based weighted averaging.

Focusing on enhancing DNN inference performance in heterogeneous edge computing, Dai et al. [45] address the cooperative DNN inference (MCDI) problem to maximize system throughput through optimized device placement and model partitioning. Their method comprises the evolutionary device placement algorithm, which employs a variant of particle swarm optimization to determine optimal device pipeline stages, and the adaptive model partitioning algorithm, applying dynamic programming for efficient intra- and inter-layer partitioning. The proposed algorithm enhances the throughput by approximately 19%

to 42% and decreases the inference time by up to 9% compared to existing competitive solutions, while also achieving significantly lower running times in large-scale scenarios.

6.2. Device–Edge Collaboration

The paradigm of device–edge collaboration involves an integrated operational framework wherein end-user devices, ranging from sensors to smartphones, collaborate with edge computing entities, including but not limited to edge servers and gateways, through local area networks. This model inherits advantages akin to those found in device–device collaboration, notably minimal latency and the facilitation of local data processing. Beyond these benefits, device–edge collaboration enables dynamic task designation tailored to the specific characteristics of the engaged computational resources, while also providing flexible scalability concerning the computational nodes involved.

Within the framework of device–edge collaboration, the principal technologies that are frequently applied include model partitioning [129–132], task offloading [130,133], task partitioning [35], and communication optimization [134,135]. Table 9 provides an overview of relevant works in device–edge collaboration, detailing their supported models, hardware platforms, main technologies, key metrics, and code availability.

Table 9. Device–edge collaboration.

Work	Supported Models	Platforms	Main Technology	Key Metrics	Code Availability
[129]	AlexNet VGG16 ResNet-18 GoogLeNet	NVIDIA Jetson TX2 NVIDIA Titan Xp	Model partitioning	Latency Storage Accuracy	✗
[35]	ResNet-50 VGG-16	NVIDIA RTX 2080Ti NVIDIA GTX 1080Ti NVIDIA Jetson Xavier	Task partitioning	Accuracy Latency Service failure probability	✗
DisCFNN [130]	CFNN-A CFNN-V CFNN-R	Intel Xeon Platinum 8352V	Model partitioning Task offloading	Utility Success rate Server utilization rate Relative transmission data size	✗
Roulette [136]	LeNet ResNet18 ResNet50	NVIDIA A100 Intel Xeon Gold 6240	Model partitioning Differential privacy	Fairness Accuracy Attack accuracy Computing load	✗
[131]	VGG-16	NVIDIA Tesla V100	Model partitioning Data fusion	Accuracy Latency Transmission gain Communication overhead	✗
Edgent [132]	AlexNet	Intel Quad-core Processor Raspberry Pi 3	Model partitioning Early exit	Latency Accuracy Throughput	✗
LSTM-TD3 [133]	MobileNetV3-Large MobileNetV3-Small	Raspberry Pi NVIDIA GTX 960	Task offloading	Latency Accuracy Execution time	✗
MAHPPO [134]	ResNet-18 VGG-11 MobileNetV2	NVIDIA Jetson Nano	Feature compression PPO	Compression rate Latency Energy consumption	✓
[135]	VGG16	Unmentioned	Early exit Transmision decision	Accuracy Communication savings	✗
[137]	DeiT-Tiny DeiT-Small DeiT-Base	APPLE iPhone 12 NVIDIA RTX 3090	Patch selection Communication optimization	Communication cost Accuracy	✗

Li et al. [129] present an automatic tuning neural network quantization framework utilizing model partitioning. Firstly, they perform an analysis of the characteristics of different layers in DNNs. For inception networks, partition points with brother branches as candidates are excluded; in residual networks, those with shortcut connections are omitted. Then, the framework performs auto-tuning partition based on the candidate rule, selecting candidate points, testing their performance, and finding the optimal partition for mixed-precision neural network inference between the edge and the devices. The proposed auto-tuning neural network quantization framework for collaborative inference reduces

model storage on mobile devices by up to 99.97%, with a trivial accuracy loss of usually less than 1%; in some cases, it can achieve a speed-up of up to 1.7×.

Focusing on accelerating CNN inference in collaborative edge computing networks for time-critical IoT applications, Li et al. [35] introduce RFS to partition CNN tasks among ESs without compromising model accuracy. They also propose a fused-layer parallelization technique that merges multiple convolutional layers into a single block to reduce computation time and communication overhead. Utilizing dynamic programming for fused-layer parallelization, they optimize CNN model partitioning and design a low-complexity search algorithm to select the optimal subset of ESs considering computational heterogeneity. The proposed method can accelerate the inference of VGG-16 up to 73% compared with the pre-trained model and reduce the communication time by about 90% at a given transmission rate, while ensuring a service reliability of over 99.999% under strict service deadlines.

Hu et al. [130] present DisCFNN, a device–edge collaborative inference framework for convolutional fuzzy neural networks (CFNN) in big-data-driven IoT systems. It features an offline-designed CFNN structure with an optimized fuzzy layer to reduce computing load, where polynomial regression estimates the calculation amount of this layer. Edge servers use DRL for computing resource allocation and CFNN partitioning, incorporating a "resource pricing" concept to address the resource allocation challenge. A many-to-one matching game is employed for task offloading, aiming to balance profit between IoT customers and service providers. Compared with baselines, DisCFNN shows improvements in various key metrics, such as achieving a total server utility 0.31–0.42 times higher than DisCFNN-NoPar when the number of customers rises from 3 to 15, and maintaining a higher server utilization rate and fairness in latency and energy consumption while having a smaller relative transmission data size.

Li et al. [136] propose Roulette to tackle challenges in device–edge collaborative inference for deep learning classification, specifically addressing non-i.i.d. data and privacy leakage. Roulette retrains the front-end DNN on-device with local data, customizing the model to local characteristics. For privacy protection, it treats data ground truth as private and trains the DNN to function as both a feature extractor and an encryptor by randomly remapping data–label relationships, supplemented by a differential privacy mechanism that adds noise to intermediate representations. In a situation with severe non-i.i.d., Roulette improves the inference accuracy by 21% averaged over benchmarks and reduces the accuracy of discrimination attacks to almost random guessing levels.

Focusing on the challenges of deep learning at the edge, Palena et al. [131] propose leveraging the spatial correlation of data from multiple end devices to enhance multi-view classification efficiency. They categorize these schemes into centralized inference (CI) and ensemble inference (EI). CI includes non-selective CI, where views are transmitted to a central controller for classification, and selective CI (SCI), with variants SCI-E and SCI-CH, which decide transmission based on cosine similarity or color histograms, respectively. EI comprises non-selective EI, where local single-view classifications are aggregated by a central controller, and selective EI (SEI) with SEI-E and SEI-CH variants, utilizing embeddings and color histograms to determine local inference necessity. These approaches aim to balance prediction quality with resource consumption. The proposed selective collaborative schemes can reduce communication overhead by 18–74% compared to centralized inference while maintaining an inference accuracy well above 90% and also show different trade-offs in other metrics like inference latency.

Li et al. [132] present Edgent, a framework for on-demand DNN collaborative inference via device–edge synergy. Edgent employs DNN partitioning to adaptively distribute computation between mobile devices and edge servers based on available bandwidth, thus reducing execution latency. It also incorporates DNN right-sizing, which accelerates

inference through an early-exit mechanism at a suitable layer, balancing accuracy and latency. For different network environments, Edgent uses regression-based models in static settings to determine optimal configurations and an online change-point detection algorithm in dynamic settings to generate the best execution plan, ensuring high inference accuracy while meeting latency constraints. In a static bandwidth environment where the latency requirement is fixed at 1000 ms and the bandwidth increases from 50 kbps to 1.5 Mbps, Edgent can change the optimal exit point to increase the accuracy, and the inference latency first drops sharply and then rises abruptly; in a dynamic bandwidth environment, compared with the static configurator, the dynamic configurator of Edgent can achieve a 10 FPS higher throughput at a CDF of 0.6, which significantly improves the co-inference efficiency.

Focusing on the DNN inference task offloading problem in device–edge computing, Cui et al. [133] formulate a multi-objective optimization problem to minimize average delay and maximize average inference accuracy. Their proposed LSTM-TD3 algorithm integrates LSTM for capturing long-term environmental information from time-series data with the twin delayed deep deterministic policy gradient (TD3) algorithm. TD3, featuring an actor–critic architecture, facilitates efficient exploration in the high-dimensional action space of task offloading decisions. This combined approach maps environmental states to weight matrices, guiding optimal offloading decisions to enhance system performance. The LSTM-TD3 algorithm reduces the average inference delay by up to 21.6% compared with the random offloading algorithm and has a better accuracy performance, with its average accuracy being closer to the upper bound and higher than that of the random and TD3 (without LSTM) algorithms in different network sizes.

In a multi-user-equipment (UE) and single-edge-server scenario for DNN inference, Hao et al. [134] propose a framework that employs a lightweight autoencoder-based feature compression method, combining autoencoders and quantization to reduce feature size efficiently. This approach adapts to various DNNs while minimizing UE resource consumption. The framework formulates the multi-user collaborative inference problem as a Markov decision process, enabling optimization through reinforcement learning. Their MAHPPO algorithm, with an actor–critic structure, handles complex action spaces and enhances robustness in uncertain multi-agent environments. Compared to the all-local-inference strategy, the proposed method can reduce the inference latency by 56% and save 72% of energy consumption when the number of UEs is three, significantly enhancing the key metrics of multi-agent collaborative inference.

Jankowski et al. [135] propose a device–edge method that splits a DNN for image classification. Intermediate features are compressed using joint source-channel coding and transmitted over an additive white Gaussian noise channel. Early exit layers at the DNN's split point enable early predictions, while a transmission-decision neural network evaluates these predictions alongside channel state information to decide whether to accept the early exit or transmit data for further processing. This approach reduces communication needs by avoiding unnecessary feature transmissions and adapts to changing wireless conditions, balancing classification accuracy with communication costs. In the wireless edge collaborative inference system for image classification, applying the proposed TD mechanisms can achieve significant communication savings (nearly 45% when SNR $= 0$ dB) while maintaining comparable or better classification accuracy compared to using only early exit or final exit.

Im et al. [137] propose a framework for efficient collaborative inference using pre-trained ViT models. It uses a lightweight ViT on the device and a complex ViT on the edge. The approach employs attention-aware patch selection and entropy-aware transmission. The former transmits only essential image patches based on attention scores; the latter

decides local or edge server processing by evaluating min-entropy from the device model's output. This reduces communication overhead and server computational load with minimal accuracy loss, providing an efficient device–edge inference solution. The proposed collaborative inference framework can reduce communication overhead by 68% with only a minimal loss (from 81.8% to 80.84%) in accuracy compared to the server model on the ImageNet dataset.

6.3. Device–Cloud Collaboration

Device–cloud collaboration denotes a cooperative framework in which terminal devices collaborate with cloud servers through remote connections including LAN or the Internet to execute inference tasks. Generally, terminal devices are tasked with operations demanding high real-time responsiveness, whereas the cloud undertakes complex calculations and large-scale data analyses. This approach boasts significant dynamic adaptability, merging the low-latency advantages of terminal devices with the superior computational capabilities of the cloud. It facilitates extensive data processing and comprehensive optimization, delivering efficient and scalable inference functionalities by distributing computational workloads. Consequently, this model is especially apt for environments necessitating a balance between real-time performance and computational complexity.

In the context of device–cloud collaboration, several key technologies are commonly employed, including model partitioning [43,138–140], task offloading [141–143], model compression [43,139–141], and scheme searching [139,144], among others. Deep reinforcement learning (DRL) techniques are sometimes integrated into these methods [44] to enhance performance. Table 10 summarizes various efforts in device–cloud collaboration, listing the supported models, hardware platforms, main technologies utilized, critical indicators, and source code availability.

Zhang et al. [138] propose EdgeShard, a framework for efficient LLM inference in collaborative edge computing, to address the issues of long latency and high bandwidth costs in cloud-based LLM deployment and resource limitation on end devices. EdgeShard divides LLMs into shards and distributes them across end devices and cloud servers. The framework adopts profiling, scheduling optimization and proper KV-cache management to optimize LLM inference, and formulates problems for minimizing latency and maximizing throughput. Dynamic programming algorithms are designed for both, taking into account device and network characteristics. EdgeShard reduces the inference latency of popular Llama2 serial models by up to 50% and improves the throughput by 2× compared to the state of the art, enabling efficient LLM inference in collaborative edge computing.

Zhang et al. [44] introduce DVFO, a DVFS-enabled learning-based collaborative inference framework for DNNs that co-optimizes CPU, GPU, and memory frequencies on devices along with the proportion of feature maps offloaded to cloud servers. DVFO employs a thinking-while-moving concurrent mechanism within DRL to expedite policy inference in dynamic environments. It also uses a spatial-channel attention mechanism to assess feature map importance, guiding the offloading process to alleviate network bottlenecks and enhance inference efficiency without compromising accuracy. DVFO significantly reduces energy consumption by 33% on average and achieves a 28.6–59.1% end-to-end latency reduction while maintaining accuracy within 1% loss on average for various DNN models.

Table 10. Device-cloud collaboration.

Work	Supported Models	Platforms	Main Technology	Key Metrics	Code Availability
EdgeShard [138]	Llama2-7B Llama2-13B Llama2-70B	NVIDIA Jetson Orin NVIDIA Jetson Orin NX NVIDIA RTX 3090	Model partitioning Pipeline execution optimization	Latency Throughput	✗
DVFO [44]	EfficientNetB0 ViT-B16 ResNet-18 Inception-v4 MobileNet-v2 YOLOv3-Tiny RetinaNet DeepSpeech	NVIDIA Jetson Nano NVIDIA Jetson TX2 NVIDIA Jetson Xavier NX NVIDIA Orin NX NVIDIA AGX Orin NVIDIA RTX 3080	Dynamic voltage frequency scaling Deep reinforcement learning	Latency Energy Accuracy	✗
Opt-CoInfer [139]	VGG-16	Raspberry Pi 4B NVIDIA Tesla V100	Model partitioning Model compression Optimal scheme searching	Latency Accuracy	✓
[43]	VGG16 ResNet18 MobileNetV1 MobileNetV2	Raspberry Pi 3B NVIDIA RTX 3080Ti	Model partitioning Network pruning Feature compression	Latency Accuracy	✗
[140]	CenterNet	NVIDIA Jetson Nano NVIDIA RTX 3090 NVIDIA GTX 1060	Model pruning Model partitioning	Accuracy Latency	✗
EARLIN [145]	DenseNet ResNet34 ResNet44 VGG16	Intel Core i7 9750H NVIDIA Tesla K80	Early exit	Accuracy Latency	✗
Hybrid SD [141]	Stable Diffusion v1.4 BK-SDM-Small BK-SDM-Tiny OursTiny	NVIDIA A100 NVIDIA Tesla V100 APPLE iPhone 15 Pro	Model pruning Task offloading	Quality	✗
DRAX [144]	AlexNet VGG11 ResNet34	Intel Stratix IV GX FPGA Intel Xeon Silver 4114 Intel Neural Compute Stick 2 NVIDIA Jetson Nano Google Edge TPU	Heuristic approximation	Energy consumption Accuracy	✓
[142]	Llama2-70B-chat Llama2-7B-chat TinyLlama-1.1B	Unmentioned	Task offloading	Accuracy Cost	✗
PerLLM [143]	Llama2-33B Llama2-7B Llama3-8B Yi-6B Yi-9B	Intel Xeon Silver 4214R NVIDIA A100	Task offloading Resource allocation	Latency Throughput Energy consumption	✗

Zhang et al. [139] introduce Opt-CoInfer for achieving optimal collaborative inference in CNNs, targeting fast and accurate performance. The framework formulates the problem as identifying the best collaboration scheme to meet accuracy or latency requirements, employing layer-wise partitioning and CNN compression techniques like feature map pruning and quantization. It operates through three processes: pre-processing to obtain latency profiles and initialize settings, an optimal collaboration scheme searching process that iteratively evaluates promising schemes using a Gaussian process model and updates the local optimum to narrow the feasible set, and a collaborative inference process that executes inference based on the determined optimal scheme. Opt-CoInfer achieves up to 3.49 times faster inference with the same accuracy requirement and as low as 37.41% accuracy loss with the same latency requirement compared to single-end and state-of-the-art CI approaches.

Li et al. [43] introduce a device–cloud collaborative inference framework for DNN deployment, integrating model splitting, network pruning, and feature coding. It features a sparsity-aware feature bias minimization pruning method for the device-side model to tackle underfitting and over-sparsity issues. For reduced communication overhead, task-oriented asymmetric feature coding is utilized, enabling efficient data reconstruction on the cloud side while significantly cutting computation at the encoder. Compared with traditional cloud–edge collaborative inference frameworks, the proposed method can reduce end-to-end latency by 82–84% with less than 1% accuracy loss, and also shows superiority in computation–communication trade-off and low-bandwidth scenarios.

Zhang et al. [140] focus on reducing the end-to-end inference latency of face detection models on resource-constrained devices. They adopt a two-step acceleration strategy for the CenterNet model. In the first step, the model is pruned based on the L_1 weight regularization, and then global fine-tuning is performed. Moreover, deconvolution pruning is carried out to avoid redundancy. In the second step, the optimizer searches for the best split point of the model based on the current bandwidth of end devices and acceptable latency requirements by establishing a relationship mapping table related to different split points and device computing resources. Compared with the state-of-the-art object detection model Blazeface, the proposed method achieves a 62.12% reduction in inference latency in the first strategy, and with a two-step speedup strategy, its latency is only 26.5% of the baseline when the bandwidth is 500 kbps.

Nimi et al. [145] introduce EARLIN, a lightweight method for early out-of-distribution (OOD) detection in collaborative inference, which extracts and selects informative feature maps from a pre-trained CNN's shallow layers using indexed-pooling and max-pooling. EARLIN defines a distance function based on ID data and sets a threshold to identify OOD samples. In the collaborative setup, it partitions the model, placing an OOD-detection part on devices and the rest on the cloud. Detected ID samples are sent to the cloud for classification; OOD samples are flagged, thus saving resources. EARLIN enables early OOD detection with minimal computation, operates as an external module without retraining, and avoids needing OOD samples for tuning, ensuring reliable detection. EARLIN significantly improves the true negative rate at 95% true positive rate, with values reaching up to 99.96% in some cases, and also shows higher detection accuracy and AUROC compared to previous approaches, reducing the detection error and enhancing the overall performance of out-of-distribution detection.

Yan et al. [141] introduce Hybrid SD, a framework for edge–cloud collaborative inference with stable diffusion models. It splits the process: a cloud model handles early semantic planning, while an edge model refines visual details, reducing cloud load and utilizing edge resources efficiently. The model is pruned based on layer-significance scores for edge deployment, and a lightweight VAE is trained via encoder distillation and advanced decoder strategies. This approach lowers cloud costs and enables high-quality image generation on edge devices. Hybrid SD reduces cloud cost by 66% while maintaining competitive visual quality in stable diffusion models, achieving a remarkable balance between cost efficiency and performance.

Das et al. [144] introduce DRAX for energy-efficient collaborative DNN inference in multiview 3D classification using MVCNNs. It optimizes CNN segmentation with the accuracy-over-energy metric and assesses view significance using Shannon's entropy. DRAX employs significance-aware group assignment to reduce design space and significance-aware approximation for tailored node approximation, improving energy–accuracy trade-offs. It also adjusts approximations based on node energy levels and uses a gradient-descent heuristic to optimize settings, enhancing energy efficiency for resource-constrained devices in real-time applications. The DRAX method achieves significant energy savings, with 2.6×–8× reduction for minimal (<1%) application-level quality loss and up to 34× savings for ~2.5% quality loss, along with 1.5×–5× and 2.4×–22.6× speed-up in edge inference latency for <1% and <2.5% quality degradation, respectively, and reduces the total communication cost by 6.2×–40× and 5.9×–31× for a <1% and <2.5% quality loss bound, respectively.

Hao et al. [142] introduce a dynamic token-level device–cloud collaboration for LLMs, using a small language model (SLM) such as TinyLlama on devices. During inference, the SLM generates tokens, and their probability distribution from the cloud-side LLM decides if they meet a predefined threshold. Tokens exceeding the threshold are kept; others are

replaced by the LLM. This method uses SLMs for drafting and LLMs for verification, balancing quality and cost. It reduces LLM calls and device–cloud communications compared to traditional methods, applies to existing models without modifications, and enhances adaptability and efficiency in collaborative LLM inference. The proposed method can achieve LLM-comparable quality with only 25.8%, 31.2%, and 27.2% of the LLM's cost on GSM8K, HumanEval, and NaturalQuestion tasks, respectively.

Yang et al. [143] introduce PerLLM to address resource management in large-scale LLM services. It formulates device–cloud collaborative inference scheduling as a multi-objective optimization problem, targeting minimal energy cost under processing time, bandwidth, and computing capacity constraints. By formulating this as a combinatorial multi-armed bandit problem, with actions as service-to-server assignments and states as server resources, they propose the constraint satisfaction upper confidence bound algorithm, which integrates constraint satisfaction into reward calculations, making decisions that maximize rewards while meeting constraints and adapting to dynamic changes. PerLLM boosts the success rate of meeting service processing time requirements to over 97%, reduces average processing time, increases throughput by more than 1.6 times, and cuts energy costs by over 50% compared to baseline methods.

6.4. Device–Edge–Cloud Collaboration

Device–edge–cloud collaboration refers to a hierarchical network approach where terminal devices, edge computing nodes, and cloud servers work together to accomplish inference tasks. Typically, terminal devices handle real-time tasks, edge nodes provide low-latency local computation, and the cloud is responsible for complex analysis and global optimization. This relatively complex structure enables complementary advantages of real-time performance and computational capabilities across these three different scales of devices. By employing layered computation and dynamic task allocation, it can achieve efficient, flexible, and scalable inference capabilities. This model is particularly suitable for complex application scenarios that require a balance between real-time performance, complex computing, and global optimization.

Within the framework of device–cloud collaboration, the principal technologies that are frequently applied encompass model partitioning [146–149], computation offloading [149–153], and occasionally, strategies like reinforcement learning [148,149] and communication optimization [33] are integrated to further improve the performance of these collaborative approaches. Table 11 provides an overview of pertinent works in device–edge–cloud collaboration, detailing the supported models, hardware platforms, core technologies employed, key metrics, and source code availability.

Zhang et al. [33] introduce a Cloud-RAN-based approach for collaborative edge AI inference. Edge devices capture real-time data and extract feature vectors, aggregated by remote radio heads using over-the-air computation (AirComp) to reduce noise. They propose a joint optimization of transmit precoding, receive beamforming, and quantization error control to maximize discriminant gain—a metric for inference accuracy. The non-convex problem is solved using variable transformation, successive convex approximation, and alternating optimization, improving resource allocation and inference performance. The proposed method can achieve up to about 20–30% higher inference accuracy than baselines in datasets like human motion and Fashion MNIST, across various fronthaul capacities and energy constraints.

Liu et al. [146] propose an adaptive DNN inference acceleration framework for end–edge–cloud computing to reduce latency in resource-constrained environments. It features neural-network-based latency prediction for accurate execution time estimation and a two-point partitioning algorithm that divides DNN computations into data-intensive,

hybrid, and computation-intensive blocks. This enables efficient distribution across devices, achieving better latency balancing than traditional methods and significantly reducing inference latency by leveraging the strengths of end, edge, and cloud resources. The proposed method improves the prediction accuracy of the latency prediction model by about 72.31% on average compared with four baseline approaches and reduces the end-to-end latency by about 20.81% on average against six baseline approaches under three wireless networks.

Table 11. Device–edge–cloud collaboration.

Work	Supported Models	Platforms	Main Technology	Key Metrics	Code Availability
[33]	SVM MLP	Unmentioned	Communication optimization	Accuracy	✗
[146]	AlexNet ResNet-34 MobileNetV1	Huawei Nova 7 Pro NVIDIA Max250 NVIDIA GTX 1080Ti x 3	Model partitioning	Latency	✗
CRIME [150]	CoVe a one-layer LSTM	ARM Cortex-A53 NVIDIA Jetson TX2 NVIDIA Titan Xp	Task offloading	Latency Energy consumption	✗
C-NMT [154]	BiLSTM GRU RNN "MarianMT" Transformer	NVIDIA Jetson TX2 NVIDIA Titan Xp	Task estimation Linear mapping	Execution time	✗
[151]	HyperLPR YOLOv5 MTCNN	Intel Core i7 10510U NVIDIA RTX 2080Ti	Task offloading	Latency Throughput Traffic Device utilization	✗
CNNPC [147]	MobileNet-V2 ResNet-18 SSD-VGG16	Snapdragon 845 Snapdragon 710 NVIDIA Jetson TX2 NVIDIA Tesla P100	Model partitioning Model compression	Latency Accuracy Compression rate	✓
MCIA [148]	ResNet-56	Intel Core i7 9700K	Model partitioning Deep reinforcement learning	Latency Accuracy	✗
[149]	AlexNet MobileNet-v2 GoogLeNet	Unmentioned	Task offloading Model partitioning Resource allocation PPO	Latency Throughput	✗
[152]	BERT-Base-uncased BERT-Large-uncased BERTweet	NVIDIA RTX 4090	Task offloading Early exit	Accuracy Execution time	✗
EosDNN [153]	AlexNet VGG GoogleNet ResNet	Unmentioned	Computation offloading	Latency	✓

Pagliari et al. [150] introduce CRIME, a method for collaborative RNN inference modeled as a directed acyclic graph. Each device decides to compute locally or offload based on estimates of processing speeds and network conditions, accounting for input length and computational resources. Devices maintain models for execution and transmission times, with mechanisms for updating these models to adapt to changes in network speed and load. This approach enables efficient resource utilization in collaborative RNN inference. Experiments on several RNNs and datasets show that CRIME can reduce the execution time (or end-node energy) by more than 25 percent compared to any single-device approach.

Chen et al. [154] propose C-NMT, a collaborative inference framework for neural machine translation (NMT) that addresses decoder execution time variability by estimating output length from input sentence length. C-NMT models the total execution time and decides whether to perform inference at the edge or in the cloud based on these estimates. It is the first to apply collaborative inference to seq2seq problems in NMT, reducing latency by leveraging the relationship between input and output lengths and making intelligent device selection decisions. This approach provides a more efficient solution compared to non-collaborative and naive methods, enhancing NMT performance. Experiments show C-NMT cuts the total execution time of 100,000 translation requests by as much as 26% (DE-EN), 44% (FR-EN), and 36% (EN-ZH) relative to pure edge or cloud strategies.

Gao and Zhang [151] present a semantics-driven cloud–edge approach for video inference, using license plate detection as an example. The method splits the process into semantics extraction by edge servers and recognition tasks offloaded based on load conditions. Edge servers extract visual semantics from video frames, reducing data transfer needs, and perform recognition if within their capacity; otherwise, tasks are distributed to neighboring edges or the cloud. This strategy reduces latency, improves throughput by utilizing edge and cloud resources, and decreases traffic volume by transferring only essential data. The proposed semantics-driven cloud–edge collaborative approach reduces end-to-end latency by up to 4/5, increases average throughput up to 5× (reaching ~9 FPS with some algorithms), and decreases cloud–edge traffic by about 50% compared to traditional processing methods.

Yang et al. [147] introduce CNNPC, a method for efficient CNN inference in device–edge–cloud systems that reduces computation latency and data uploading costs. It profiles each CNN layer's performance across devices and applies compression techniques like identical channel pruning and 8-bit quantization to minimize data size and transmission latency. By solving sub-problems of minimizing latency under accuracy constraints and maximizing accuracy under latency constraints, CNNPC uses an efficient algorithm for optimal partitioning and compression strategy selection. Compared with state-of-the-art single-end and collaborative approaches, CNNPC achieves up to 1.6 and 5.6 times faster collaborative inference with as low as 4.30% and 6.48% communications respectively, and requires only 0.1% of the actual compression operations that the traversal method requires when determining the optimal strategy, all without obvious accuracy loss.

Qi et al. [148] propose the MCIA framework to address DNN inference challenges such as complex edge environments, diverse service requirements, and resource allocation. MCIA models the problem as a mixed-integer multi-dimensional optimization and trains multiple DNNs with varying compression scales in the cloud, deploying them to end devices and edge servers. A DRL-based algorithm makes end-to-end decisions on model version, partition, and resource allocation by interacting with the environment through a defined state space, action space, and reward function balancing accuracy and latency. The MCIA method can achieve up to 95% inference accuracy when the task demands high accuracy, with a latency of about 13 ms when favoring low latency, and outperforms other methods in terms of average reward across different scenarios, such as in homogeneous and heterogeneous end-device environments with varying delay weights, bandwidths, and numbers of end devices.

In the context of cloud–edge–end systems for DNN tasks, Tian et al. [149] propose a novel approach to address collaborative inference and task offloading challenges. They introduce the optional partition point compression algorithm, which transforms the model into a chain structure to identify and reduce partition points based on layer output features, simplifying high-quality partition selection. For decision making in dynamic environments, they develop the reinforcement-learning-based collaborative inference optimization (RLCIO) algorithm, aimed at minimizing average end-to-end latency. RLCIO uses a PPO-like training architecture but decouples resource allocation via the edge computing resource allocation algorithm, reducing decision variables and improving convergence. The RLCIO algorithm reduces the average end-to-end latency of system tasks by 72% and improves the system throughput by 3.5× in the best case compared to five related schemes.

Zhang et al. [152] propose a multi-level collaborative inference system for next-generation networks, aimed at managing GAI's high computational demands on resource-limited devices. The system uses distributed deployment with larger models on cloud servers and smaller models on user devices. It implements confidence-based task offloading and attention-based pruning to reduce delays, alongside an early exit mechanism to

save computational resources by terminating inference early. This approach effectively balances computational needs and inference quality, surpassing traditional methods. The proposed multi-level collaborative inference system can reduce inference time by up to 17% without sacrificing inference accuracy compared to existing work, thus achieving a significant improvement in the key metrics of inference latency and quality.

Xue et al. [153] propose EosDNN, an efficient offloading scheme for DNN inference in device–edge–cloud environments, addressing the limitations of mobile devices' computing power. EosDNN uses PSO-GA for task distribution, enabling multi-task parallelism and optimizing layer distribution across servers to minimize migration delay. It also introduces the layer merge uploading algorithm (LMU) to combine adjacent DNN layers, reducing partition granularity and improving query performance. By enhancing both migration and uploading processes, EosDNN provides a comprehensive solution that surpasses traditional methods in DNNs management. Compared with other methods, the EosDNN offloading scheme reduces the maximum distribution ratio of DNN partitions, and for AlexNet and VGG models, it optimizes the SSD of the layer merge uploading algorithm by 45.16% and obtains a lower SSD, respectively, while also achieving a lower migration delay under different algorithms.

6.5. Summary

In this section, we explore collaborative inference and categorize it into four types: device–device, device–edge, device–cloud, and device–edge–cloud collaboration. Device–device collaboration enables resource-constrained devices to cooperate, offering low latency and privacy benefits via model partitioning and task assignment. Device–edge collaboration pairs end devices with edge nodes for localized data processing, using model partitioning and task offloading to minimize latency. Device–cloud collaboration leverages terminal low-latency operations and cloud computing power through similar methods, optimizing performance. The device–edge–cloud approach integrates all three elements, employing model partitioning and computation offloading for efficient and flexible inference. Overall, these collaborative inference models and their associated techniques play a vital role in meeting the diverse demands of different application scenarios.

7. Future Research Opportunities

7.1. The Interpretability of NAS

NAS aims to replace the experience-based design method through automated search, aiming to design structures with comparable or even better inference performance than manually designed models. It integrates actual hardware and application requirements to search for efficient model structures under multi-index constraints. Initially, large-scale evolution [155] utilized evolutionary algorithms (EA) as the search strategy for NAS, but the vast search space posed certain limitations on its search performance. Subsequently, NASNet [156] adopted ideas from the experience-based design method and introduced a modular search space. While this approach significantly reduces the search cost, it may also overlook some optimization opportunities. Thus, balancing the search space and search efficiency remains a critical research direction. Considering the high cost of NAS, we suggest optimizing it from the following three directions: (1) employing path-level pruning techniques to reduce unnecessary architecture search space; (2) using binary parameter methods instead of full-precision structural parameters to decrease GPU memory usage and computational requirements; (3) combining gradient-based methods with reinforcement learning algorithms to further enhance search efficiency while ensuring search quality. Moreover, the current theoretical support for NAS is inadequate. Specifically, although the widely used parameter sharing strategy [157] can effectively improve search efficiency,

related studies have indicated that this strategy might lead to suboptimal architectures [158]. Therefore, further exploration of the underlying principles governing the performance-dominant aspects of the search process is necessary.

7.2. Multi-Optimized Overlay Compression

Pruning, quantization, and sparsification acceleration are commonly used model compression techniques. Initially, research works applied these methods separately to compress neural network models. As model sizes and application demands continue to grow, one straightforward approach is to stack these methods. This superposition compression can significantly reduce the model size, but it also leads to a notable decrease in accuracy. To address this issue, refs. [24–26] propose combining multiple compression methods using a rotation matrix. However, during implementation, because different compression methods focus on distinct optimization objectives, joint compression increases the design complexity compared to simple stacking (for instance, quantization primarily emphasizes the uniformity of data distribution, while pruning focuses more on data sparsity). Therefore, designing an effective joint optimization algorithm is crucial. For example, we can develop a multi-objective evolutionary algorithm (MOEA) that simultaneously considers multiple optimization objectives such as model size, computational efficiency, and inference accuracy. This new algorithm can integrate reinforcement learning mechanisms, use policy networks to guide the search process, and automatically explore the optimal pruning strategy, sparsity pattern, and quantization level.

7.3. Graphics and Computing Joint Compilation Optimization

Graph optimization reduces redundant calculations and enhances memory access efficiency through techniques like operator fusion and splitting. Code generation technology integrates the optimized computation graph with scheduling optimization to efficiently produce executable programs for target hardware. Current approaches treat computation graph optimization and code generation scheduling optimization as two separate modules. Specifically, existing deep learning frameworks consider the operators in the computation graph as basic abstract units and map these operators to the corresponding operator libraries of the hardware platform. While this operator abstraction is intuitive, it makes it challenging to explore joint optimization opportunities between operators. Therefore, finding a fine-grained abstraction method to achieve graph-computation joint optimization is critical. Although [114,159] have decomposed single operators into fine-grained parallel subtasks to implement graph-computation joint compilation optimization and perceive the relationships between operators for global analysis and optimization, these approaches require significant manual optimization efforts on the target hardware. Consequently, further investigation into the scalability of graph-computation joint compilation optimization is warranted for future research.

7.4. Intelligent Task Allocation and Combination Optimization

The continuous increase in model size and the slowdown of Moore's Law have led to a trend toward collaborative model inference. Although multiple hardware platforms offer relatively abundant computing and storage resources, model inference must consider more optimization factors compared to single-hardware scenarios. Combining multiple optimization schemes (such as model partitioning and computation–communication trade-offs) has expanded the optimization space, making it challenging to find the optimal solution. First, in collaborative inference scenarios, addressing the typical model partitioning problem is essential to avoid the "straggler" phenomenon caused by uneven task distribution [155,160,161]. Secondly, reducing data dependencies between model layers is necessary to improve the efficiency of dataflow synchronization. Finally, finding a unified

optimization solution [162] that accommodates multiple optimizations is crucial. When optimization sub-schemes are stacked, they create a vast search space, making it vital to efficiently and accurately locate the optimal parallel solution within this space to reduce the cost of collaborative inference. Moreover, privacy protection in edge-based collaborative inference is equally important, particularly in edge–cloud collaborative reasoning scenarios. Besides traditional encryption methods, modifying model weight parameters for privacy protection has emerged as a new research direction. For instance, model protection [163] modifies less than 1% of the model weights by combining important parameters with carefully designed random values, making it difficult for adversaries to identify noise parameters, restore the original model, or retrain it. Additionally, privacy-aware DNN partitioning strategies can be employed. He et al. [164] suggest incorporating privacy factors when selecting split points in a collaborative system, recommending placing at least one fully connected layer on the edge device. This provides new approaches for enhancing inference data security and offers valuable insights for designing more secure collaborative inference systems.

7.5. Technology Applications and Potential Multidisciplinary Collaborations

In the rapidly developing fields of science and technology, the application of technology and its potential for multidisciplinary cooperation have become particularly important. Technologies such as lightweight model design, model compression, optimized compilation toolchains, and collaborative inference not only provide solutions to complex AI tasks in resource-constrained environments but also foster deep integration and innovation across multiple disciplines. These technologies are transforming industries including industrial automation, smart cities, and medical devices by improving efficiency, reducing costs, and enhancing system performance.

Lightweight model design enhances the efficiency and responsiveness of vehicle perception systems, thereby improving safety by reducing computing latency. By combining research from computer science and electronic engineering, the collaboration between hardware and software can be further optimized to achieve more efficient model deployment, such as matching algorithm implementations to specific hardware platforms to maximize performance and minimize resource consumption. Materials science explores new materials and manufacturing processes, enhancing the performance of edge devices and better supporting the operation of lightweight models. Beyond improving inference speed, model compression technology also reduces data transmission, enabling faster real-time data analysis in 5G networks and decreasing model sizes in smart home systems, which improves energy efficiency and extends device lifespan. Additionally, energy management research indicates that model compression significantly reduces energy consumption, promoting the development of a sustainable smart ecosystem. For virtual reality (VR) and augmented reality (AR) applications, an optimized compilation toolchain greatly improves rendering speed and interactive experiences, delivering smoother and more immersive outcomes. Collaboration between software engineering and hardware engineering enables automatic adjustment of models to different processor architectures through precise hardware-aware optimization, maximizing computing efficiency. Collaborative inference technology optimizes traffic management systems and enhances public safety levels through the collaboration of distributed nodes in smart cities. In telemedicine, local preliminary health assessments are combined with in-depth cloud-based analysis to accelerate diagnoses and improve the quality and accessibility of medical services.

In summary, by integrating knowledge and technology from multiple fields such as computer science, electronic engineering, materials science, communication engineering,

and sociology, multidisciplinary cooperation will continue to drive the application of new technologies and accelerate societal progress and development.

8. Conclusions

EI is developing rapidly to meet the needs of intelligent applications in resource-constrained scenarios. This paper explores four key optimization directions: model design, model compression, compilation toolchain, and collaborative inference. Lightweight model design, whether achieved through experience or NAS, simplifies models used for EI. Model compression reduces complexity, the compilation toolchain optimizes model execution, and collaborative inference leverages multiple devices for enhanced performance. Although significant progress has been made in various research directions, challenges for efficient EI persist. These include the high computational cost of NAS, the need for improvements in multi-optimization overlay compression, the necessity for graph-computation joint compilation to better address scalability, and the requirement for collaborative inference task allocation to adapt to diverse application needs. This paper discusses future research directions, including NAS with lower training costs, more effective multi-optimization overlay compression techniques, advancements in graph-computation joint compilation, and smarter task allocation strategies. It is hoped that this survey will contribute to making edge-based services across different fields more intelligent, efficient, and privacy preserving.

Author Contributions: All authors conducted reference research, wrote the original manuscript, and revised the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the National Key Research and Development Program of China (No. 2022YFB4501600).

Data Availability Statement: No new data generated in this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kiran, B.R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A.A.; Yogamani, S.; Pérez, P. Deep reinforcement learning for autonomous driving: A survey. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 4909–4926. [CrossRef]
2. Dogan, Ü.; Edelbrunner, J.; Iossifidis, I. Autonomous driving: A comparison of machine learning techniques by means of the prediction of lane change behavior. In Proceedings of the IEEE International Conference on Robotics and Biomimetics, Karon Beach, Thailand, 7–11 December 2011; pp. 1837–1843.
3. Bachute, M.R.; Subhedar, J.M. Autonomous driving architectures: Insights of machine learning and deep learning algorithms. *Mach. Learn. Appl.* **2021**, *6*, 100164. [CrossRef]
4. Bhavsar, K.A.; Singla, J.; Al-Otaibi, Y.D.; Song, O.Y.; Zikria, Y.B.; Bashir, A.K. Medical diagnosis using machine learning: A statistical review. *Comput. Mater. Contin.* **2021**, *67*, 107–125. [CrossRef]
5. Bhavsar, K.A.; Abugabah, A.; Singla, J.; AlZubi, A.A.; Bashir, A.K. A comprehensive review on medical diagnosis using machine learning. *Comput. Mater. Contin.* **2021**, *67*, 1997. [CrossRef]
6. Richens, J.G.; Lee, C.M.; Johri, S. Improving the accuracy of medical diagnosis with causal machine learning. *Nat. Commun.* **2020**, *11*, 3923. [CrossRef]
7. Alzoubi, A. Machine learning for intelligent energy consumption in smart homes. *Int. J. Comput. Inf. Manuf. (IJCIM)* **2022**, *2*. [CrossRef]
8. Javed, A.R.; Fahad, L.G.; Farhan, A.A.; Abbas, S.; Srivastava, G.; Parizi, R.M.; Khan, M.S. Automated cognitive health assessment in smart homes using machine learning. *Sustain. Cities Soc.* **2021**, *65*, 102572. [CrossRef]
9. Priyadarshini, I.; Sahu, S.; Kumar, R.; Taniar, D. A machine-learning ensemble model for predicting energy consumption in smart homes. *Internet Things* **2022**, *20*, 100636. [CrossRef]
10. Ullah, Z.; Al-Turjman, F.; Mostarda, L.; Gagliardi, R. Applications of artificial intelligence and machine learning in smart cities. *Comput. Commun.* **2020**, *154*, 313–323. [CrossRef]
11. França, R.P.; Monteiro, A.C.B.; Arthur, R.; Iano, Y. An overview of the machine learning applied in smart cities. In *Smart Cities: A Data Analytics Perspective*; Springer: Cham, Switzerland, 2021; pp. 91–111.

12. Prawiyogi, A.G.; Purnama, S.; Meria, L. Smart cities using machine learning and intelligent applications. *Int. Trans. Artif. Intell.* **2022**, *1*, 102–116. [CrossRef]
13. Xu, G.; Hao, Z.; Luo, Y.; Hu, H.; An, J.; Mao, S. DeViT: Decomposing vision transformers for collaborative inference in edge devices. *IEEE Trans. Mob. Comput.* **2023**, 23, 5917–5932. [CrossRef]
14. Li, N.; Iosifidis, A.; Zhang, Q. Distributed deep learning inference acceleration using seamless collaboration in edge computing. In Proceedings of the ICC 2022-IEEE International Conference on Communications, Seoul, Republic of Korea, 16–20 May 2022; pp. 3667–3672.
15. Dhar, A.C.; Roy, A.; Biswas, S.; Islam, B. Studying the security threats of partially processed deep neural inference data in an iot device. In Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems, Boston, MA, USA, 6–9 November 2022; pp. 845–846.
16. Ryu, J.; Zheng, Y.; Gao, Y.; Abuadbba, A.; Kim, J.; Won, D.; Nepal, S.; Kim, H.; Wang, C. Can differential privacy practically protect collaborative deep learning inference for IoT? *Wirel. Netw.* **2024**, *30*, 4713–4733.
17. Baccour, E.; Erbad, A.; Mohamed, A.; Hamdi, M.; Guizani, M. Distprivacy: Privacy-aware distributed deep neural networks in iot surveillance systems. In Proceedings of the GLOBECOM 2020—2020 IEEE Global Communications Conference, Taiwan, China, 7–11 December 2020; pp. 1–6.
18. Zhang, R.; Jiang, H.; Geng, J.; Tian, F.; Ma, Y.; Wang, H. A high-performance dataflow-centric optimization framework for deep learning inference on the edge. *J. Syst. Archit.* **2024**, *152*, 103180. [CrossRef]
19. Zhou, A.; Yang, J.; Qi, Y.; Qiao, T.; Shi, Y.; Duan, C.; Zhao, W.; Hu, C. HGNAS: Hardware-Aware Graph Neural Architecture Search for Edge Devices. *IEEE Trans. Comput.* **2024**, *73*, 2693–2707.
20. Cai, H.; Zhu, L.; Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv* **2018**, arXiv:1812.00332.
21. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
22. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
23. Yang, C.; Zhao, P.; Li, Y.; Niu, W.; Guan, J.; Tang, H.; Qin, M.; Ren, B.; Lin, X.; Wang, Y. Pruning parameterization with bi-level optimization for efficient semantic segmentation on the edge. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, 17–24 June 2023; pp. 15402–15412.
24. Anonymous. RotPruner: Large language model pruning in rotated space. *arXiv* **2024**, arXiv:2410.09426.
25. Xiao, G.; Lin, J.; Seznec, M.; Demouth, J.; Han, S. FLATQUANT: Flatness Matters for LLM Quantization. *arXiv* **2024**, arXiv:2410.09426.
26. Liu, Z.; Zhao, C.; Fedorov, I.; Soran, B.; Choudhary, D.; Krishnamoorthi, R.; Chandra, V.; Tian, Y.; Blankevoort, T. SpinQuant: LLM Quantization with Learned Rotations. *arXiv* **2024**, arXiv:2405.16406.
27. Hsu, O.; Strange, M.; Sharma, R.; Won, J.; Olukotun, K.; Emer, J.S.; Horowitz, M.A.; Kjølstad, F. The sparse abstract machine. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Vancouver, BC, Canada, 25–29 March 2023; Volume 3, pp. 710–726.
28. Shi, Y.; Tang, A.; Niu, L.; Zhou, R. Sparse optimization guided pruning for neural networks. *Neurocomputing* **2024**, *574*, 127280. [CrossRef]
29. Wang, H.; Zhai, J.; Gao, M.; Ma, Z.; Tang, S.; Zheng, L.; Li, Y.; Rong, K.; Chen, Y.; Jia, Z. PET: Optimizing tensor programs with partially equivalent transformations and automated corrections. In Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), Online, 14–16 July 2021; pp. 37–54.
30. Jia, Z.; Padon, O.; Thomas, J.; Warszawski, T.; Zaharia, M.; Aiken, A. TASO: Optimizing deep learning computation with automatic generation of graph substitutions. In Proceedings of the 27th ACM Symposium on Operating Systems Principles, Porto, Portugal, 27–30 October 2019; pp. 47–62.
31. Chen, T.; Moreau, T.; Jiang, Z.; Zheng, L.; Yan, E.; Shen, H.; Cowan, M.; Wang, L.; Hu, Y.; Ceze, L.; et al. TVM: An automated End-to-End optimizing compiler for deep learning. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, USA, 8–10 October 2018; pp. 578–594.
32. Zhao, J.; Li, B.; Nie, W.; Geng, Z.; Zhang, R.; Gao, X.; Cheng, B.; Wu, C.; Cheng, Y.; Li, Z.; et al. AKG: Automatic kernel generation for neural processing units using polyhedral transformations. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Online, 20–25 June 2021; pp. 1233–1248.
33. Zhang, P.; Wen, D.; Zhu, G.; Chen, Q.; Han, K.; Shi, Y. Collaborative Edge AI Inference over Cloud-RAN. *IEEE Trans. Commun.* **2024**, *72*, 5641–5656.
34. Xu, Z.; Zhang, P.; Li, C.; Zhu, H.; Xu, G.; Sun, C. A Collaborative Inference Algorithm in Low-Earth-Orbit Satellite Network for Unmanned Aerial Vehicle. *Drones* **2023**, *7*, 575. [CrossRef]
35. Li, N.; Iosifidis, A.; Zhang, Q. Collaborative edge computing for distributed cnn inference acceleration using receptive field-based segmentation. *Comput. Netw.* **2022**, *214*, 109150. [CrossRef]

36. OpenAI. Available online: https://openai.com/ (accessed on 1 March 2025).
37. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Proceedings of the Advances in Neural Information Processing Systems 25 (NIPS 2012), Lake Tahoe, NV, USA, 3–6 December 2012; Volume 25.
38. Fang, C.; Guo, S.; Wu, W.; Lin, J.; Wang, Z.; Hsu, M.K.; Liu, L. An efficient hardware accelerator for sparse transformer neural networks. In Proceedings of the 2022 IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 27 May–1 June 2022; pp. 2670–2674.
39. Yao, Z.; Yazdani Aminabadi, R.; Zhang, M.; Wu, X.; Li, C.; He, Y. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 27168–27183.
40. Wu, D.; Tang, Q.; Zhao, Y.; Zhang, M.; Fu, Y.; Zhang, D. Easyquant: Post-training quantization via scale optimization. *arXiv* **2020**, arXiv:2006.16669.
41. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A system for Large-Scale machine learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
42. ONNX. Available online: https://github.com/onnx/onnx (accessed on 1 March 2025).
43. Li, M.; Zhang, X.; Guo, J.; Li, F. Cloud–Edge Collaborative Inference with Network Pruning. *Electronics* **2023**, *12*, 3598. [CrossRef]
44. Zhang, Z.; Zhao, Y.; Li, H.; Lin, C.; Liu, J. DVFO: Learning-Based DVFS for Energy-Efficient Edge-Cloud Collaborative Inference. *IEEE Trans. Mob. Comput.* **2024**, *23*, 9042–9059.
45. Dai, P.; Han, B.; Li, K.; Xu, X.; Xing, H.; Liu, K. Joint Optimization of Device Placement and Model Partitioning for Cooperative DNN Inference in Heterogeneous Edge Computing. *IEEE Trans. Mob. Comput.* **2024**, *24*, 210–226.
46. Zhou, Y.; Chen, S.; Wang, Y.; Huan, W. Review of research on lightweight convolutional neural networks. In Proceedings of the 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 12–14 June 2020; pp. 1713–1720.
47. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
48. Huang, G.; Liu, S.; Van der Maaten, L.; Weinberger, K.Q. Condensenet: An efficient densenet using learned group convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 2752–2761.
49. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 6848–6856.
50. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 10–15 June 2019; pp. 6105–6114.
51. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
52. Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; Le, Q.V. Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach Convention & Entertainment Center, Long Beach, CA, USA, 15–20 June 2019; pp. 2820–2828.
53. Liu, H.; Simonyan, K.; Yang, Y. Darts: Differentiable architecture search. *arXiv* **2018**, arXiv:1806.09055.
54. Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.J.; Fei-Fei, L.; Yuille, A.; Huang, J.; Murphy, K. Progressive neural architecture search. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 19–34.
55. Cai, H.; Yang, J.; Zhang, W.; Han, S.; Yu, Y. Path-level network transformation for efficient architecture search. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 678–687.
56. Real, E.; Aggarwal, A.; Huang, Y.; Le, Q.V. Regularized evolution for image classifier architecture search. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 4780–4789.
57. Lyu, B.; Yuan, H.; Lu, L.; Zhang, Y. Resource-constrained neural architecture search on edge devices. *IEEE Trans. Netw. Sci. Eng.* **2021**, *9*, 134–142. [CrossRef]
58. Luo, X.; Liu, D.; Huai, S.; Kong, H.; Chen, H.; Liu, W. Designing efficient DNNs via hardware-aware neural architecture search and beyond. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *41*, 1799–1812. [CrossRef]
59. Risso, M.; Burrello, A.; Conti, F.; Lamberti, L.; Chen, Y.; Benini, L.; Macii, E.; Poncino, M.; Pagliari, D.J. Lightweight neural architecture search for temporal convolutional networks at the edge. *IEEE Trans. Comput.* **2022**, *72*, 744–758. [CrossRef]
60. Akin, B.; Gupta, S.; Long, Y.; Spiridonov, A.; Wang, Z.; White, M.; Xu, H.; Zhou, P.; Zhou, Y. Searching for efficient neural architectures for on-device ML on edge TPUs. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 19–20 June 2022; pp. 2667–2676.
61. Frankle, J.; Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv* **2018**, arXiv:1803.03635.

62. Cheng, Y.; Wang, D.; Zhou, P.; Zhang, T. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Process. Mag.* **2018**, *35*, 126–136. [CrossRef]
63. Li, G.; Ma, X.; Wang, X.; Yue, H.; Li, J.; Liu, L.; Feng, X.; Xue, J. Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning. *J. Syst. Archit.* **2022**, *124*, 102431. [CrossRef]
64. Wang, H.; Ling, P.; Fan, X.; Tu, T.; Zheng, J.; Chen, H.; Jin, Y.; Chen, E. All-in-one hardware-oriented model compression for efficient multi-hardware deployment. *IEEE Trans. Circuits Syst. Video Technol.* **2024**, *34*, 12345–12359. [CrossRef]
65. Goyal, V.; Das, R.; Bertacco, V. Hardware-friendly user-specific machine learning for edge devices. *ACM Trans. Embed. Comput. Syst. (TECS)* **2022**, *21*, 1–29. [CrossRef]
66. Jiang, Y.; Wang, S.; Valls, V.; Ko, B.J.; Lee, W.H.; Leung, K.K.; Tassiulas, L. Model pruning enables efficient federated learning on edge devices. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *34*, 10374–10386. [CrossRef]
67. Kong, H.; Liu, D.; Luo, X.; Huai, S.; Subramaniam, R.; Makaya, C.; Lin, Q.; Liu, W. Towards Efficient Convolutional Neural Network for Embedded Hardware via Multi-Dimensional Pruning. In Proceedings of the 2023 60th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 9–13 July 2023; pp. 1–6.
68. Yu, T.; Wu, B.; Chen, K.; Yan, C.; Liu, W. Data stream oriented fine-grained sparse CNN accelerator with efficient unstructured pruning strategy. In Proceedings of the Great Lakes Symposium on VLSI 2022, Orange County, CA, USA, 6–8 June 2022; pp. 243–248.
69. Yu, Z.; Wang, Z.; Li, Y.; Gao, R.; Zhou, X.; Bommu, S.R.; Zhao, Y.; Lin, Y. Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting. In Proceedings of the 61st ACM/IEEE Design Automation Conference, San Francisco, CA, USA, 23–27 June 2024; pp. 1–6.
70. Yin, R.; Kim, Y.; Li, Y.; Moitra, A.; Satpute, N.; Hambitzer, A.; Panda, P. Workload-balanced pruning for sparse spiking neural networks. *IEEE Trans. Emerg. Top. Comput. Intell.* **2024**, *8*, 2897–2907.
71. Eccles, B.J.; Wong, L.; Varghese, B. Rapid deployment of dnns for edge computing via structured pruning at initialization. In Proceedings of the 2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Philadelphia, PA, USA, 6–9 May 2024; pp. 317–326.
72. Joardar, B.K.; Doppa, J.R.; Li, H.; Chakrabarty, K.; Pande, P.P. ReaLPrune: ReRAM crossbar-aware lottery ticket pruning for CNNs. *IEEE Trans. Emerg. Top. Comput.* **2022**, *11*, 303–317.
73. Aggarwal, S.; Binici, K.; Mitra, T. CRISP: Hybrid Structured Sparsity for Class-Aware Model Pruning. In Proceedings of the 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE), Valencia, Spain, 25–27 March 2024; pp. 1–6.
74. Chou, W.C.; Huang, C.W.; Huang, J.D. Hardware-friendly progressive pruning framework for CNN model compression using universal pattern sets. In Proceedings of the 2022 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Taiwan, China, 18–21 April 2022; pp. 1–4.
75. Wang, J.; Yu, S.; Yuan, Z.; Yue, J.; Yuan, Z.; Liu, R.; Wang, Y.; Yang, H.; Li, X.; Liu, Y. PACA: A pattern pruning algorithm and channel-fused high PE utilization accelerator for CNNs. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2022**, *41*, 5043–5056.
76. Gong, Y.; Zhan, Z.; Zhao, P.; Wu, Y.; Wu, C.; Ding, C.; Jiang, W.; Qin, M.; Wang, Y. All-in-one: A highly representative dnn pruning framework for edge devices with dynamic power management. In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, San Diego, CA, USA, 30 October–3 November 2022; pp. 1–9.
77. Gao, Y.; Zhang, B.; Qi, X.; So, H.K.H. Dpacs: Hardware accelerated dynamic neural network pruning through algorithm-architecture co-design. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, San Diego, CA, USA, 25–29 March 2023; Volume 2, pp. 237–251.
78. Sui, X.; Lv, Q.; Zhi, L.; Zhu, B.; Yang, Y.; Zhang, Y.; Tan, Z. A hardware-friendly high-precision CNN pruning method and its FPGA implementation. *Sensors* **2023**, *23*, 824. [CrossRef]
79. Wang, Y.; Qin, Y.; Liu, L.; Wei, S.; Yin, S. SWPU: A 126.04 TFLOPS/W edge-device sparse DNN training processor with dynamic sub-structured weight pruning. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2022**, *69*, 4014–4027. [CrossRef]
80. Gale, T.; Elsen, E.; Hooker, S. The state of sparsity in deep neural networks. *arXiv* **2019**, arXiv:1902.09574.
81. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv* **2015**, arXiv:1510.00149.
82. Zhou, A.; Ma, Y.; Zhu, J.; Liu, J.; Zhang, Z.; Yuan, K.; Sun, W.; Li, H. Learning n: M fine-grained structured sparse neural networks from scratch. *arXiv* **2021**, arXiv:2102.04010.
83. Liu, Z.; Wang, Y.; Han, K.; Zhang, W.; Ma, S.; Gao, W. Post-training quantization for vision transformer. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 28092–28103.
84. Shang, Y.; Yuan, Z.; Xie, B.; Wu, B.; Yan, Y. Post-training quantization on diffusion models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 17–24 June 2023; pp. 1972–1981.
85. Liu, F.; Zhao, W.; He, Z.; Wang, Y.; Wang, Z.; Dai, C.; Liang, X.; Jiang, L. Improving neural network efficiency via post-training quantization with adaptive floating-point. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Online, 10–17 October 2021; pp. 5281–5290.

86. Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.M.; Wang, W.C.; Xiao, G.; Dang, X.; Gan, C.; Han, S. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proc. Mach. Learn. Syst.* **2024**, *6*, 87–100.
87. Shen, X.; Dong, P.; Lu, L.; Kong, Z.; Li, Z.; Lin, M.; Wu, C.; Wang, Y. Agile-quant: Activation-guided quantization for faster inference of LLMs on the edge. In Proceedings of the the AAAI Conference on Artificial Intelligence, British Columbia, BC, Canada, 20–27 February 2024; Volume 38, pp. 18944–18951.
88. Liu, Z.; Oguz, B.; Zhao, C.; Chang, E.; Stock, P.; Mehdad, Y.; Shi, Y.; Krishnamoorthi, R.; Chandra, V. Llm-qat: Data-free quantization aware training for large language models. *arXiv* **2023**, arXiv:2305.17888.
89. Zhou, Q.; Guo, S.; Qu, Z.; Guo, J.; Xu, Z.; Zhang, J.; Guo, T.; Luo, B.; Zhou, J. Octo: INT8 training with loss-aware compensation and backward quantization for tiny on-device learning. In Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC 21), Online, 14–16 July 2021; pp. 177–191.
90. Kim, D.; Lee, J.; Ham, B. Distance-aware quantization. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Online, 10–17 October 2021; pp. 5271–5280.
91. Matinizadeh, S.; Mohammadhassani, A.; Pacik-Nelson, N.; Polykretisl, I.; Mishra, A.; Shackleford, J.; Kandasamy, N.; Gallo, E.; Das, A. A fully-configurable digital spiking neuromorphic hardware design with variable quantization and mixed precision. In Proceedings of the 2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS), Springfield, MA, USA, 11–14 August 2024; pp. 937–941.
92. Liu, X.; Wang, T.; Yang, J.; Tang, C.; Lv, J. MPQ-YOLO: Ultra low mixed-precision quantization of YOLO for edge devices deployment. *Neurocomputing* **2024**, *574*, 127210.
93. Gao, T.; Guo, L.; Zhao, S.; Xu, P.; Yang, Y.; Liu, X.; Wang, S.; Zhu, S.; Zhou, D. QuantNAS: Quantization-aware Neural Architecture Search For Efficient Deployment On Mobile Device. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 17–18 June 2024; pp. 1704–1713.
94. Lin, J.; Zhu, L.; Chen, W.M.; Wang, W.C.; Gan, C.; Han, S. On-device training under 256 KB memory. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 22941–22954.
95. Parashar, A.; Rhu, M.; Mukkara, A.; Puglielli, A.; Venkatesan, R.; Khailany, B.; Emer, J.; Keckler, S.W.; Dally, W.J. SCNN: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Comput. Archit. News* **2017**, *45*, 27–40. [CrossRef]
96. Krishna, A.; Nudurupati, S.R.; Chandana, D.; Dwivedi, P.; van Schaik, A.; Mehendale, M.; Thakur, C.S. Raman: A re-configurable and sparse tinyML accelerator for inference on edge. *IEEE Internet Things J.* **2024**, *11*, 24831–24845.
97. Zhang, J.F.; Lee, C.E.; Liu, C.; Shao, Y.S.; Keckler, S.W.; Zhang, Z. SNAP: An efficient sparse neural acceleration processor for unstructured sparse deep neural network inference. *IEEE J. Solid-State Circuits* **2020**, *56*, 636–647.
98. Gondimalla, A.; Chesnut, N.; Thottethodi, M.; Vijaykumar, T. SparTen: A sparse tensor accelerator for convolutional neural networks. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, Columbus, OH, USA, 12–16 October 2019; pp. 151–165.
99. Meng, J.; Venkataramanaiah, S.K.; Zhou, C.; Hansen, P.; Whatmough, P.; Seo, J.s. Fixyfpga: Efficient fpga accelerator for deep neural networks with high element-wise sparsity and without external memory access. In Proceedings of the 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), Dresden, Germany, 30 August–3 September 2021; pp. 9–16.
100. Vasireddy, P.; Kavi, K.; Mehta, G. Sparse-t: Hardware accelerator thread for unstructured sparse data processing. In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, San Diego, CA, USA, 29 October–3 November 2022; pp. 1–8.
101. Zhang, S.; Du, Z.; Zhang, L.; Lan, H.; Liu, S.; Li, L.; Guo, Q.; Chen, T.; Chen, Y. Cambricon-X: An accelerator for sparse neural networks. In Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taiwan, China, 15–19 October 2016; pp. 1–12.
102. Zhou, X.; Du, Z.; Guo, Q.; Liu, S.; Liu, C.; Wang, C.; Zhou, X.; Li, L.; Chen, T.; Chen, Y. Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach. In Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Cambridge, UK, 20–24 October 2018; pp. 15–28.
103. Kjolstad, F.; Chou, S.; Lugato, D.; Kamil, S.; Amarasinghe, S. Taco: A tool to generate tensor algebra kernels. In Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), Ulm, Germany, 30 October–3 November 2017; pp. 943–948.
104. Zheng, N.; Lin, B.; Zhang, Q.; Ma, L.; Yang, Y.; Yang, F.; Wang, Y.; Yang, M.; Zhou, L. SparTA: Deep-Learning Model Sparsity via Tensor-with-Sparsity-Attribute. In Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), Vancouver, BC, Canada, 11–13 July 2022; pp. 213–232.
105. Ye, Z.; Lai, R.; Shao, J.; Chen, T.; Ceze, L. Sparsetir: Composable abstractions for sparse compilation in deep learning. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Vancouver, BC, Canada, 25–29 March 2023; Volume 3, pp. 660–678.

106. Tian, R.; Guo, L.; Li, J.; Ren, B.; Kestor, G. A high performance sparse tensor algebra compiler in MLIR. In Proceedings of the 2021 IEEE/ACM 7th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC), San Diego, CA, USA, 14 November 2021; pp. 27–38.
107. Liu, R.; Leng, Y.; Tian, S.; Hu, S.; Chen, C.F.; Yao, S. DynaSpa: Exploiting Spatial Sparsity for Efficient Dynamic DNN Inference on Devices. In Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems, Hangzhou, China, 4–7 November 2024; pp. 422–435.
108. Zhang, G.; Hsu, O.; Kjolstad, F. Compilation of modular and general sparse workspaces. *Proc. ACM Program. Lang.* **2024**, *8*, 1213–1238. [CrossRef]
109. Xia, H.; Zheng, Z.; Li, Y.; Zhuang, D.; Zhou, Z.; Qiu, X.; Li, Y.; Lin, W.; Song, S.L. Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. *arXiv* **2023**, arXiv:2309.10285.
110. Jia, Z.; Thomas, J.; Warszawski, T.; Gao, M.; Zaharia, M.; Aiken, A. Optimizing DNN computation with relaxed graph substitutions. *Proc. Mach. Learn. Syst.* **2019**, *1*, 27–39.
111. Zheng, L.; Wang, H.; Zhai, J.; Hu, M.; Ma, Z.; Wang, T.; Huang, S.; Miao, X.; Tang, S.; Huang, K.; et al. EINNET: Optimizing tensor programs with Derivation-Based transformations. In Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23), San Diego, CA, USA, 10–12 July 2023; pp. 739–755.
112. Niu, W.; Guan, J.; Wang, Y.; Agrawal, G.; Ren, B. 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, Online, 20–25 June 2021; pp. 883–898.
113. Zheng, S.; Chen, S.; Song, P.; Chen, R.; Li, X.; Yan, S.; Lin, D.; Leng, J.; Liang, Y. Chimera: An analytical optimizing framework for effective compute-intensive operators fusion. In Proceedings of the 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Montreal, QC, Canada, 25 February–1 March 2023; pp. 1113–1126.
114. Shi, Y.; Yang, Z.; Xue, J.; Ma, L.; Xia, Y.; Miao, Z.; Guo, Y.; Yang, F.; Zhou, L. Welder: Scheduling deep learning memory access via tile-graph. In Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23), San Diego, CA, USA, 10–12 July 2023; pp. 701–718.
115. Wang, F.; Shen, M. Automatic Kernel Generation for Large Language Models on Deep Learning Accelerators. In Proceedings of the 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), San Francisco, CA, USA, 28 October–2 November 2023; pp. 1–9.
116. Meng, J.; Zhuang, C.; Chen, P.; Wahib, M.; Schmidt, B.; Wang, X.; Lan, H.; Wu, D.; Deng, M.; Wei, Y.; et al. Automatic generation of high-performance convolution kernels on ARM CPUs for deep learning. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 2885–2899. [CrossRef]
117. Danopoulos, D.; Kachris, C.; Soudris, D. Automatic generation of fpga kernels from open format cnn models. In Proceedings of the 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Online, 3–6 May 2020; p. 237.
118. Fu, Q.; Huang, H.H. Automatic generation of high-performance inference kernels for graph neural networks on multi-core systems. In Proceedings of the 50th International Conference on Parallel Processing, Lemont, IL, USA, 9–12 August 2021; pp. 1–11.
119. Zhao, X.; Chen, Z.; Shi, Y.; Wen, M.; Zhang, C. Automatic End-to-End Joint Optimization for Kernel Compilation on DSPs. In Proceedings of the 2023 60th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 9–13 July 2023; pp. 1–6.
120. Zheng, L.; Jia, C.; Sun, M.; Wu, Z.; Yu, C.H.; Haj-Ali, A.; Wang, Y.; Yang, J.; Zhuo, D.; Sen, K.; et al. Ansor: Generating High-Performance tensor programs for deep learning. In Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), Seattle, WA, USA, 4–6 November 2020; pp. 863–879.
121. Ma, L.; Xie, Z.; Yang, Z.; Xue, J.; Miao, Y.; Cui, W.; Hu, W.; Yang, F.; Zhang, L.; Zhou, L. Rammer: Enabling holistic deep learning compiler optimizations with rTasks. In Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), Seattle, WA, USA, 4–6 November 2020; pp. 881–897.
122. Zheng, B.; Jiang, Z.; Yu, C.H.; Shen, H.; Fromm, J.; Liu, Y.; Wang, Y.; Ceze, L.; Chen, T.; Pekhimenko, G. DietCode: Automatic optimization for dynamic tensor programs. *Proc. Mach. Learn. Syst.* **2022**, *4*, 848–863.
123. Li, N.; Iosifidis, A.; Zhang, Q. Receptive Field-based Segmentation for Distributed CNN Inference Acceleration in Collaborative Edge Computing. In Proceedings of the ICC 2022-IEEE International Conference on Communications, Busan, Republic of Korea, 16–20 May 2022; pp. 4281–4286.
124. Ye, S.; Du, J.; Zeng, L.; Ou, W.; Chu, X.; Lu, Y.; Chen, X. Galaxy: A Resource-Efficient Collaborative Edge AI System for In-situ Transformer Inference. *arXiv* **2024**, arXiv:2405.17245.
125. Dong, Z.; Li, N.; Iosifidis, A.; Zhang, Q. Design and prototyping distributed CNN inference acceleration in edge computing. In Proceedings of the European Wireless 2022; 27th European Wireless Conference, VDE, Oslo, Norway, 19–21 September 2022; pp. 1–6.

126. Chen, Y.; Luo, T.; Fang, W.; Xiong, N.N. Edgeci: Distributed workload assignment and model partitioning for cnn inference on edge clusters. *ACM Trans. Internet Technol.* **2024**, *24*, 1–24. [CrossRef]
127. Malka, M.; Farhan, E.; Morgenstern, H.; Shlezinger, N. Decentralized low-latency collaborative inference via ensembles on the edge. *IEEE Trans. Wirel. Commun.* **2024**, *24*, 598–614. [CrossRef]
128. Kumazawa, S.; Yu, J.; Kawamura, K.; Van Chu, T.; Motomura, M. Toward Improving Ensemble-Based Collaborative Inference at the Edge. *IEEE Access* **2024**, *12*, 6926–6940. [CrossRef]
129. Li, G.; Liu, L.; Wang, X.; Dong, X.; Zhao, P.; Feng, X. Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, 4–7 October 2018*; Proceedings, Part I 27; Springer: Cham, Switzerland, 2018; pp. 402–411.
130. Hu, Y.; Xu, X.; Duan, L.; Bilal, M.; Wang, Q.; Dou, W. End-Edge Collaborative Inference of Convolutional Fuzzy Neural Networks for Big Data-Driven Internet of Things. *IEEE Trans. Fuzzy Syst.* **2024**, *33*, 203–217. [CrossRef]
131. Palena, M.; Cerquitelli, T.; Chiasserini, C.F. Edge-device collaborative computing for multi-view classification. *Comput. Netw.* **2024**, *254*, 110823. [CrossRef]
132. Li, E.; Zeng, L.; Zhou, Z.; Chen, X. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Trans. Wirel. Commun.* **2019**, *19*, 447–457. [CrossRef]
133. Cui, E.; Yang, D.; Wang, H.; Zhang, W. Learning-based deep neural network inference task offloading in multi-device and multi-server collaborative edge computing. *Trans. Emerg. Telecommun. Technol.* **2022**, *33*, e4485. [CrossRef]
134. Hao, Z.; Xu, G.; Luo, Y.; Hu, H.; An, J.; Mao, S. Multi-agent collaborative inference via dnn decoupling: Intermediate feature compression and edge learning. *IEEE Trans. Mob. Comput.* **2022**, *22*, 6041–6055. [CrossRef]
135. Jankowski, M.; Gündüz, D.; Mikolajczyk, K. Adaptive Early Exiting for Collaborative Inference over Noisy Wireless Channels. In Proceedings of the 2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN), Stockholm, Sweden, 5–8 May 2024; pp. 126–131.
136. Li, J.; Liao, G.; Chen, L.; Chen, X. Roulette: A Semantic Privacy-Preserving Device-Edge Collaborative Inference Framework for Deep Learning Classification Tasks. *IEEE Trans. Mob. Comput.* **2023**, *23*, 5494–5510.
137. Im, J.; Kwon, N.; Park, T.; Woo, J.; Lee, J.; Kim, Y. Attention-Aware Semantic Communications for Collaborative Inference. *IEEE Internet Things J.* **2024**, *11*, 37008–37020. [CrossRef]
138. Zhang, M.; Cao, J.; Shen, X.; Cui, Z. EdgeShard: Efficient LLM Inference via Collaborative Edge Computing. *arXiv* **2024**, arXiv:2405.14371.
139. Zhang, Z.; Yu, H.; Wang, F. Opt-CoInfer: Optimal collaborative inference across IoT and cloud for fast and accurate CNN inference. *J. King Saud Univ.-Comput. Inf. Sci.* **2023**, *35*, 438–448.
140. Zhang, W.; Zhou, H.; Mo, J.; Zhen, C.; Ji, M. Accelerated Inference of Face Detection under Edge-Cloud Collaboration. *Appl. Sci.* **2022**, *12*, 8424. [CrossRef]
141. Yan, C.; Liu, S.; Liu, H.; Peng, X.; Wang, X.; Chen, F.; Fu, L.; Mei, X. Hybrid sd: Edge-cloud collaborative inference for stable diffusion models. *arXiv* **2024**, arXiv:2408.06646.
142. Hao, Z.; Jiang, H.; Jiang, S.; Ren, J.; Cao, T. Hybrid slm and llm for edge-cloud collaborative inference. In Proceedings of the Workshop on Edge and Mobile Foundation Models, Minato-ku Tokyo, Japan, 3–7 June 2024; pp. 36–41.
143. Yang, Z.; Yang, Y.; Zhao, C.; Guo, Q.; He, W.; Ji, W. Perllm: Personalized inference scheduling with edge-cloud collaboration for diverse llm services. *arXiv* **2024**, arXiv:2405.14636.
144. Das, A.; Ghosh, S.K.; Raha, A.; Raghunathan, V. Toward energy-efficient collaborative inference using multisystem approximations. *IEEE Internet Things J.* **2024**, *11*, 17989–18004.
145. Nimi, S.T.; Arefeen, A.; Uddin, Y.S.; Lee, Y. Earlin: Early out-of-distribution detection for resource-efficient collaborative inference. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, 13–17 September 2021*; Proceedings, Part I 21; Springer: Cham, Switzerland, 2021; pp. 635–651.
146. Liu, G.; Dai, F.; Xu, X.; Fu, X.; Dou, W.; Kumar, N.; Bilal, M. An adaptive DNN inference acceleration framework with end–edge–cloud collaborative computing. *Future Gener. Comput. Syst.* **2023**, *140*, 422–435.
147. Yang, S.; Zhang, Z.; Zhao, C.; Song, X.; Guo, S.; Li, H. CNNPC: End-edge-cloud collaborative CNN inference with joint model partition and compression. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 4039–4056.
148. Qi, H.; Ren, F.; Wang, L.; Jiang, P.; Wan, S.; Deng, X. Multi-compression scale DNN inference acceleration based on cloud-edge-end collaboration. *ACM Trans. Embed. Comput. Syst.* **2024**, *23*, 1–25. [CrossRef]
149. Tian, J.; Li, X.; Qin, X. Reinforcement Learning Based Collaborative Inference and Task Offloading Optimization for Cloud-Edge-End Systems. In Proceedings of the 2024 International Joint Conference on Neural Networks (IJCNN), Yokohama, Japan, 30 June–5 July 2024; pp. 1–8.
150. Pagliari, D.J.; Chiaro, R.; Macii, E.; Poncino, M. Crime: Input-dependent collaborative inference for recurrent neural networks. *IEEE Trans. Comput.* **2020**, *70*, 1626–1639.

151. Gao, Y.; Zhang, B. Semantics-Driven Cloud-Edge Collaborative Inference A Case Study of License Plate Detection. In Proceedings of the 2023 International Conference on Image Processing, Computer Vision and Machine Learning (ICICML), Chengdu, China, 3–5 November 2023; pp. 1100–1103.
152. Zhang, C.; Zheng, X.; Tao, X.; Hu, C.; Zhang, W.; Zhu, L. Distributed Collaborative Inference System in Next-Generation Networks and Communication. *IEEE Trans. Cogn. Commun. Netw.* 2025, *early access*. [CrossRef]
153. Xue, M.; Wu, H.; Li, R.; Xu, M.; Jiao, P. EosDNN: An efficient offloading scheme for DNN inference acceleration in local-edge-cloud collaborative environments. *IEEE Trans. Green Commun. Netw.* **2021**, *6*, 248–264.
154. Chen, Y.; Chiaro, R.; Maciiy, E.; Poncino, M.; Pagliari, D.J. C-NMT: A Collaborative Inference Framework for Neural Machine Translation. In Proceedings of the 2022 IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 27 May–1 June 2022; pp. 1512–1516.
155. Real, E.; Moore, S.; Selle, A.; Saxena, S.; Suematsu, Y.L.; Tan, J.; Le, Q.V.; Kurakin, A. Large-scale evolution of image classifiers. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 2902–2911.
156. Zoph, B.; Vasudevan, V.; Shlens, J.; Le, Q.V. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 8697–8710.
157. Pham, H.; Guan, M.; Zoph, B.; Le, Q.; Dean, J. Efficient neural architecture search via parameters sharing. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 4095–4104.
158. Zhang, M.; Li, H.; Pan, S.; Chang, X.; Su, S. Overcoming multi-model forgetting in one-shot NAS with diversity maximization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Online, 13–19 June 2020; pp. 7809–7818.
159. Zhang, R.; Jiang, H.; Tian, F.; Geng, J.; Li, X.; Ma, Y.; Zhu, C.; Dong, D.; Li, X.; Wang, H. Xenos: Dataflow-centric optimization to accelerate model inference on edge devices. In Proceedings of the International Conference on Database Systems for Advanced Applications, Tianjin, China, 17–20 April 2023; pp. 535–545.
160. Ho, Q.; Cipar, J.; Cui, H.; Lee, S.; Kim, J.K.; Gibbons, P.B.; Gibson, G.A.; Ganger, G.; Xing, E.P. More effective distributed ml via a stale synchronous parallel parameter server. In Proceedings of the Advances in Neural Information Processing Systems 26 (NIPS 2013), Lake Tahoe, NV, USA, 5–10 December 2013; Volume 26.
161. Cui, H.; Cipar, J.; Ho, Q.; Kim, J.K.; Lee, S.; Kumar, A.; Wei, J.; Dai, W.; Ganger, G.R.; Gibbons, P.B.; et al. Exploiting bounded staleness to speed up big data analytics. In Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14), Philadelphia, PA, USA, 19–20 June 2014; pp. 37–48.
162. Zhang, R.; Jiang, H.; Geng, J.; Ma, Y.; Zhu, C.; Wang, H. FlexPie: Accelerate Distributed Inference on Edge Devices with Flexible Combinatorial Optimization [Technical Report]. *arXiv* **2025**, arXiv:2502.15312.
163. Hou, J.; Liu, H.; Liu, Y.; Wang, Y.; Wan, P.J.; Li, X.Y. Model protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Trans. Dependable Secur. Comput.* **2021**, *19*, 4270–4284.
164. He, Z.; Zhang, T.; Lee, R.B. Attacking and protecting data privacy in edge–cloud collaborative inference systems. *IEEE Internet Things J.* **2020**, *8*, 9706–9716. [CrossRef]