From Algorithm to Hardware: A Survey on Efficient and Safe Deployment of Deep Neural Networks

Xue Geng, Zhe Wang, Chunyun Chen, Qing Xu, Kaixin Xu, Chao Jin, Manas Gupta, Xulei Yang, Zhenghua Chen, Mohamed M. Sabry Aly, Jie Lin, Min Wu, Xiaoli Li *Fellow, IEEE*

***Abstract*—Deep neural networks (DNNs) have been widely used in many artificial intelligence (AI) tasks. However, deploying them brings significant challenges due to the huge cost of memory, energy, and computation. To address these challenges, researchers have developed various model compression techniques such as model quantization and model pruning. Recently, there has been a surge in research of compression methods to achieve model efficiency while retaining the performance. Furthermore, more and more works focus on customizing the DNN hardware accelerators to better leverage the model compression techniques. In addition to efficiency, preserving security and privacy is critical for deploying DNNs. However, the vast and diverse body of related works can be overwhelming. This inspires us to conduct a comprehensive survey on recent research toward the goal of high-performance, cost-efficient, and safe deployment of DNNs. Our survey first covers the mainstream model compression techniques such as model quantization, model pruning, knowledge distillation, and optimizations of non-linear operations. We then introduce recent advances in designing hardware accelerators that can adapt to efficient model compression approaches. Additionally, we discuss how homomorphic encryption can be integrated to secure DNN deployment. Finally, we discuss several issues, such as hardware evaluation, generalization, and integration of various compression approaches. Overall, we aim to provide a big picture of efficient DNNs, from algorithm to hardware accelerators and security perspectives.**

***Index Terms*—Network compression, network quantization, network pruning, knowledge distillation, homomorphic encryption, network acceleration.**

I. Introduction

DEEP neural networks (DNNs) are currently the foundation of many modern artificial intelligence (AI) applications with superior performance. However, with millions or even billions of parameters [1], deploying DNNs to devices presents fundamental challenges due to high costs such as computational and energy costs. In particular, in addition to the performance accuracy [2], it is also essential to consider the latency or throughput, which determines if the DNN can run fast in real-time. Besides, we need to take the following hardware-related factors into account: 1) energy and power, which determines the actual running cost; 2) area, which determines the actual design cost; and 3) storage, which determines if the DNN can be stored, or run with intermediate data. Finally, when deploying DNNs, privacy is critical to ensure secure deployment. To deploy DNNs successfully, various computing architectures have been proposed to meet the above criteria. Different computing architectures might need to address various challenges when deploying DNNs. For example, cloud computing aims to centralize the computing resources and lets the cloud not directly interfere with the end user. However, it faces the latency issue and the tremendous cost of maintaining large-scale cloud centers. Meanwhile, edge computing helps reduce the latency by running DNNs at the edge (IoT devices) with direct interference from the user. It has been widely used in many areas, including autonomous driving, smart agriculture, surveillance, etc. [3]. However, edge devices provide a limited resource for computation. Model compression techniques have been widely explored to meet the above criteria among various computing architectures.

Model compression aims at the compact model to reduce the hardware cost. A smaller model requires fewer arithmetic operations and thus improves the inference speed, requires less memory bandwidth to fetch data and thus saves energy consumption, and requires less on-chip memory to store and, therefore, is less expensive. In particular, we review the mainstream approaches to compressing the DNNs, including model quantization, network pruning, and knowledge distillation, and the optimization of non-linear operations such as softmax and non-maximum suppression (NMS) as they play an essential role in deploying DNNs.

In addition, effectively deploying a compressed model poses a challenge as current hardware accelerators are optimized for uncompressed DNNs, leading to significant wastage of computation cycles and memory bandwidth when running on compressed DNN models [4]. Therefore, we need to co-design the algorithm and the hardware. This paper also presents an overview of the custom hardware accelerators, *e.g.*, Field Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs), which are designed for the compressed DNNs in terms of various network operations. In addition to efficient computation, privacy is crucial in maintaining the integrity, reliability, and availability of third-party computing, particularly in cloud computing. In the past, encryption of confidential information was the conventional approach before using the cloud model. While it may safeguard user data privacy from an untrustworthy third party, it cannot facilitate effective ciphertext computing. Conventional cryptosystems effectively protect stored data and data in transit, but they fail to secure the data while it is decrypted for processing. Therefore, many researchers have

Xue Geng, Zhe Wang, Qing Xu, Kaixin Xu, Chao Jin, Manas Gupta, Xulei Yang, Zhenghua Chen, Min Wu and Xiaoli Li are with the Institute for Infocomm Research, A*STAR, Singapore. (email: geng_xue, wang_zhe, xu_qing, Xu_Kaixin, jin_chao, manas_gupta, yang_xulei, chen_zhenghua, wumin, xlli@i2r.a-star.edu.sg).

Chunyun Chen and Mohamed M. Sabry Aly are with the Nanyang Technological University (email: chunyun001@e.ntu.edu.sg, msabry@ntu.edu.sg).

Dr. Jie Lin, Dr. Min Wu and Dr. Xiaoli Li are the corresponding authors.

TABLE I
SUMMARY OF THIS ARTICLE

Preliminaries	Brief Descriptions of Efficient and Safe DNNs		Sections
Model compression	Quantization and Entropy Encoding	Uniform and Non-uniform Quantization	Section II-A1
		Mixed-precision Quantization	Section II-A2
		Transform Quantization	Section II-A3
		Quantization for Transformers	Section II-A4
		Entropy Encoding	Section II-A5
	Network Pruning	Pruning Strategies - Before, During, and After Training	Section II-B1
		Pruning Paradigms - Unstructured, Structured, and Semi-Structured Pruning	Section II-B2
	Knowledge Distillation	Structured Knowledge	Section II-C1
		Distillation Schemes	Section II-C2
	Non-linear Operations	Non-maximum Suppression	Section II-D1
		Softmax	Section II-D2
		Activation functions	Section II-D3
	NAS & TinyML	NAS	Section II-E1
		TinyML	Section II-E2
Hardware Optimization	Accelerators on Linear Operations	Hardware Accelerating on CNNs and Transformers	Section III-B1
		Hardware Accelerating after Quantization: Mixed-Precision and Entropy Encoding	Section III-B2
		Hardware Accelerating after Pruning: Sparse Architecture	Section III-B3
	Accelerators on Non-linear Operations	Non-maximum Suppression	Section III-C1
		Softmax	Section III-C2
	Stochastic Computing Architecture	Stochastic Computing Architecture	Section III-D
Homomorphic Encryption	Homomorphic Neural Networks	Homomorphic Neural Networks	Section IV-A
		Compression on Homomorphic Neural Networks	Section IV-B
		Hardware on Homomorphic Neural Networks	Section IV-C

been exploring Fully Homomorphic Encryption (FHE). FHE is a valuable capability in distributed computation and heterogeneous networking. In this survey, we present a comprehensive review of the latest FHE techniques for DNNs and how model compression helps in these scenarios.

Few surveys exist on deep neural compression [4], [5]. However, these works mainly focus on existing models' compression and acceleration algorithms. For example, [6]–[9] mainly focused on compression algorithms of existing models, including quantization and knowledge distillation. In addition to the compression algorithms, [10] also discussed neural architecture optimization. [4] also discussed the hardware accelerators, especially on mixed-precision data and sparse architectures. In addition to these topics, our survey also discusses non-linear operation accelerators. [11] mainly focused on hardware accelerators for Recurrent Neural Networks (RNNs). [12] explored the performances of various model compression techniques on the mobile platform. [13] surveyed many works in model compression and acceleration for pre-trained language models. [14] discussed the compression algorithms for NLP tasks. [5] talked about the model compression techniques for IoT applications. Besides, some surveys exist on specific network compression techniques such as quantization [15], [16] or network pruning [17]–[19], architecture search [20]–[22], knowledge distillation [23], [24].

In summary, while some surveys focus on specific aspects of deep neural compression, there is no survey as our article on model compression from the perspective of algorithm-efficient, hardware-accelerating, and deployment-securing DNNs. These three aspects are critical for many real-world applications that require small model sizes, hardware acceleration, and secure environments to save memory, energy, and computation costs. Our contributions are summarised as follows:

- We present a comprehensive overview on neural network compression techniques. Efficient compression algorithms are essential for reducing memory and storage footprint of deep learning models. They enable faster, more energy-efficient inference, reduce bandwidth requirements, and offer adaptability to different deployment scenarios by facilitating efficient fine-tuning, thus optimizing model deployment.
- We also compile the literature on accelerating DNNs considering the hardware constraints. Hardware-aware accelerators play a crucial role in optimizing the use of specialized hardware for model inference, ensuring efficient hardware utilization, scalability, energy efficiency, and low latency.
- To secure computations on encrypted data, we review Homomorphic Encryption (HE) for applications like Homomorphic Neural Networks (HNNs) in privacy-sensitive domains. In particular, we focus on model compression and hardware accelerators, specifically for HE.
- We summarize the challenges for compressing, accelerating and securing DNNs and discuss several future directions in the promising field.

The organization of this article is shown in Table I and is summarized as follows. In Section II, we describe the model compression techniques, including quantization, pruning, knowledge distillation, and optimization on non-linear operations. Section III discusses how the hardware accelerates the compressed model. Section IV presents an overview of the privacy-preserving technique with Homomorphic Encryption (HE). Finally, Section V presents the challenges and opportunities of model compression and hardware-software co-design in real-world applications, before concluding the paper.

II. Neural Network Compression

We briefly introduce the general deep neural networks (DNNs) in Appendix A and then we introduce advanced neural network compression techniques. We classify existing approaches into four main categories:

- Quantization and Entropy Encoding: Quantization attempts to reduce the bit width of the data in a DNN, aiming at reducing the model size for memory saving and simplifying the operations for accelerating computation.

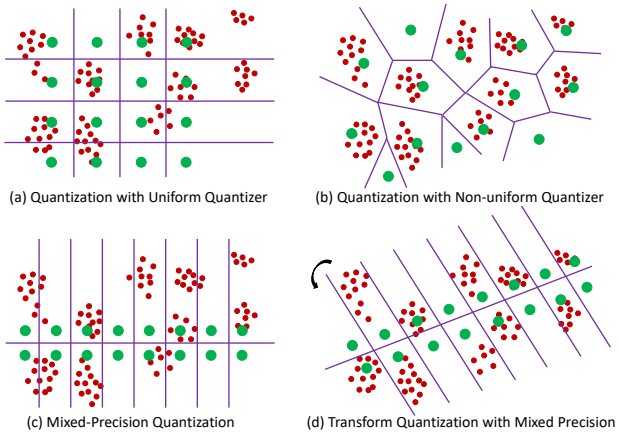

Fig. 1. (a) Quantization with uniform quantizer: the space is divided into equidistant quantization centroids. (b) Quantization with non-uniform quantizer: the space can be divided into polygonal cells based on data distribution. (c) Mixed-precision quantization: different dimensions (layers, channels, or kernels in the case of neural network quantization) are quantized with different bit widths. (d) Transform quantization with mixed precision: the space is first rotated and then quantized with mixed precision.

- Network Pruning: This category of methods aims to explore redundancy in the model parameters, eliminating non-critical and redundant parameters. This results in a more efficient model.
- Knowledge Distillation (KD): The KD methods learn a distilled model by training a more compact neural network to reproduce the output of a larger network.
- Hardware-friendly Non-linear Operations: This category of methods aims to design hardware-friendly procedures (such as linear functions) to substitute non-linear operations, shortening the hardware design cycle and speeding up inference, which is better suited for deployment on hardware platforms.

A. Quantization and Entropy Coding

Benefiting from discretizing the parameters, quantized DNNs have reduced memory cost and lower computation, as discretized parameters, after quantization, can be stored and calculated more efficiently. Moreover, by taking advantage of the peaky distribution of discretized values, entropy coding can compress the size of quantized parameters, leading higher compression ratio [25]. This subsection will review the main quantization and entropy coding approaches. In particular, first, we discuss the quantization approaches built by uniform quantizer and non-uniform quantizer. Second, we introduce specific quantization schemes developed for neural network compression, including mixed-precision and transform quantization. Finally, we discuss the entropy coding techniques.

1) Uniform Quantizer and Non-Uniform Quantizer: Quantization approaches are ususally built based on uniform quantizers and non-uniform quantizers. A uniform quantizer discretizes parameters onto equidistant quantization values as shown in Fig. 1(a), while a non-uniform quantizer maps parameters onto irregular values decided by the distribution as shown in Fig. 1(b). A non-uniform quantizer can reduce quantization error by offering greater flexibility in selecting quantization values. Nevertheless, the need for look-up tables to store discretized quantization values introduces additional implementation overheads.

Uniform quantizers are commonly employed in various studies due to their hardware-friendly nature. For instance, LQ-Nets [26] proposed the joint training of a quantized DNN and its associated quantizers, ensuring compatibility with bit operations. INQ [27] focused on converting pre-trained neural networks into low-precision versions, restricting weights to power of two or zero. DOREFA-NET [28] suggested training neural networks with low bit width weights, activations, and gradients. Additionally, numerous works utilize uniform quantizers to quantize DNNs to very low bit widths (1-bit or 2-bits). For example, Li *et al.* [29] introduced Ternary Weight Networks (TWNs) with weights constrained to 1, 0, and -1 (2-bits). Additional relevant literature on uniform quantizers can be explored in Appendix B.

On the other hand, quantization with a non-uniform quantizer involves k-means or hashing tricks and may additionally involve the Discrete Cosine Transform (DCT) and residual quantization. In particular, Gong *et al.* [30] investigated a vector quantization method to compress the weights by applying k-means clustering and conducting product quantization. HashNets [31] used a low-cost hash function to randomly group connection weights into hash buckets, and all links within the same hash bucket are quantized into a single parameter value. Chen *et al.* [32] proposed to convert weights to the frequency domain with a DCT and used hash functions to group frequency parameters into hash buckets, where parameters assigned to the same hash bucket share a single quantization value. Stock *et al.* [33] introduced a vector quantization method that aims at preserving the quality of the reconstruction of the network outputs rather than its weights, with only a set of unlabelled data needed at quantization time. APoT [34] proposed an efficient nonuniform quantization scheme for the bell-shaped and long-tailed distribution of weights and activations in neural networks. SYQ [35] introduced a quantization method to reduce information loss by learning a symmetric codebook for particular weight subgroups. Table I in Appendix B provides a comprehensive summary and comparative analysis of various quantization methods, including those utilizing both uniform and non-uniform quantizers.

2) Mixed-Precision Quantization: Mixed-precision quantization as shown in Fig. 1(c) uses distinct bit widths to quantize weights and activations of different layers, channels, or kernels. Because weights and activation in various layers, channels, or kernels exhibit different levels of sensitivity to quantization, and thus allocating different bit widths is more reasonable and can deliver higher accuracy than uniform quantization. Mixed-precision quantization can be categorized as layer-wise, channel-wise, and kernel-wise approaches based on the minimal elements used to determine the bit width.

ReLeQ [36] introduced an end-to-end deep reinforcement learning (RL) framework to automate the process of discovering quantization bit width. Alternatively, HAQ [37] leveraged RL to determine quantization bit width but employed a hardware simulator to generate direct feedback signals to the RL agent. DNAS [38] presented a differentiable neural architecture search framework to explore the hyper-parameter

TABLE II
A SUMMARY OF MIXED-PRECISION QUANTIZATION APPROACHES FOR THE RESULTS ON IMAGENET.

Methods	Bit Allocation Scheme	Model	Weight	Activation	Accuracy (Original)
ReLeQ [36]	Layer-wise	AlexNet	5 Bits MP	5 Bits MP	— (—) [a]
		MobileNet-V2	6 Bits MP	6 Bits MP	— (—) [a]
HAQ [37]	Layer-wise	MobileNet-V2	4 Bits MP	4 Bits MP	67.0% (71.8%)
		ResNet-50	2 Bits MP	32 Bits	70.6% (76.2%)
DNAS [38]	Layer-wise	ResNet-18	1.5 Bits MP	32 Bits	69.6% (71.0%)
DQ [39]	Layer-wise	MobileNet-V2	4 Bits MP	4 Bits MP	70.6% (70.2%)
		ResNet-18	4 Bits MP	4 Bits MP	70.7% (70.3%)
HAWQ [40]	Layer-wise	GoogleNet	2 Bits MP	4 Bits MP	75.5% (77.5%)
		ResNet-50	2 Bits MP	4 Bits MP	75.5% (77.4%)
		SqueezeNet	3 Bits MP	8 Bits MP	68.0% (69.4%)
ALQ [41]	Kernel-wise	ResNet-18	1 Bit MP	2 Bits MP	63.2% (69.8%)
Banner *et al.* [42]	Channel-wise	VGG	4 Bits MP	4 Bits MP	70.5% (71.6%)
		GoogleNet	4 Bits MP	4 Bits MP	66.4% (77.2%)
		ResNet-50	4 Bits MP	4 Bits MP	73.8% (76.1%)
FracBits [43]	Kernel-wise	MobileNet-V2	3 Bits MP	3 Bits MP	68.2% (71.8%)
		ResNet-18	3 Bits MP	3 Bits MP	69.8% (70.2%)
DMBQ [44]	Channel-wise	ResNet-18	1 Bit MP	2 Bits MP	63.5% (70.3%)
AutoQ [45]	Kernel-wise	ResNet-50	4 Bits MP	4 Bits MP	74.5% (74.8%)
		SqueezeNet	4 Bits MP	4 Bits MP	56.5% (56.9%)
		MobileNet-V2	4 Bits MP	4 Bits MP	70.8% (71.1%)

[a] ReLeQ [36] did not report the detailed accuracy and only reported the accuracy gap between original and compressed models (*i.e.*, -0.1% and -0.5%) on AlexNet and MobileNet-V2 respectively.

TABLE III
A SUMMARY OF COMPRESSION APPROACHES WITH ENTROPY CODING FOR THE RESULTS ON IMAGENET.

Methods	Entropy Coding	Model	Compression Ratio	Accuracy (Original)
Deep Compression [25]	Huffman Coding (FVL)	AlexNet	35×	57.2% (57.2%)
		VGG	49×	68.2% (68.5%)
Coreset [46]	Huffman Coding (FVL)	AlexNet	55×	57.1% (57.2%)
		VGG	238×	68.1% (68.9%)
DeepCABAC [47]	Arithmetic Coding (FVL)	VGG	64×	69.4% (69.4%)
		ResNet-50	17×	74.1% (76.1%)
		MobileNet-V1	8×	66.2% (70.7%)
EPR [48]	Arithmetic Coding (FVL)	ResNet-50	19×	74.0% (75.0%)
RDOC [49]	Tunstall Coding (VFL)	ResNet-50	20×	75.0% (75.5%)
		MobileNet-V2	10×	70.2% (71.0%)

space of quantization bit width. Differentiable Quantization (DQ) [39] learned quantizer parameters, such as step size and range, through training with straight-through gradients and then inferred quantization bit width based on the learned step size and range. Hessian AWare Quantization (HAWQ) [40] introduced a second-order quantization method to select the quantization bit width for each layer based on the Hessian spectrum. All the above methods deploy a layer-wise bit allocation scheme.

Compared with layer-wise approaches, channel-wise or kernel-wise approaches can achieve higher accuracy because of their fine-grained quantization. To this end, Khoram *et al.* [50] proposed Adaptive Quantization (AQ) which simplifies a trained DNN model by finding a unique, optimal precision for each network parameter such that the increase in loss is minimized. Qu *et al.* [41] proposed Adaptive Loss-aware Quantization (ALQ) which achieves an average bit width below one-bit without significant loss in inference accuracy. Meanwhile, Banner *et al.* [42] adopted channel-wise bit allocation to improve quantization precision and provided an analytic solution for quantization bit width, assuming certain distributions of parameters. FracBits [43] generalized quantization bit width to an arbitrary real number to make it differentiable and learned channel-wise bit allocation during training. Distribution-aware Multi-Bit Quantization (DMBQ) [44] proposed a loss-guided bit width allocation strategy to adjust the bit width of weights and activations channel-wisely. Lastly, AutoQ [45] presented a hierarchical deep reinforcement learning approach to find the quantization bit width of channels while simultaneously optimizing hardware metrics such as latency and energy. Table II summarizes the mixed-precision quantization approaches discussed above. In Table II, the bit width of mixed-precision quantization approaches is the average bit width across all the layers. For example, if a two-layer model quantizes one layer to 4 bits and the other one to 8 bits, then the average bit width of this mixed-precision quantization is (4+8)/2 = 6 bits. The variance in reported accuracies for CNN architectures like MobileNet-V2 across literature [39], [45] complicates the evaluation of quantization's impact on model performance. This inconsistency could be due to the differences in dataset pre-processing, training protocols, or minor architectural modifications. To accurately evaluate quantization effects, it requires a standardized approach towards reporting the performance metrics of both original and quantized models. This would facilitate a more precise isolation of quantization's effects from other influencing factors. As a result, emphasizing the relative changes in model accuracy resulting from quantization—rather than focusing solely on absolute accuracy figures—offers a more robust measure for evaluating the effectiveness of quantization techniques. Such an approach would help in better understanding the trade-offs between model efficiency and performance in the context of quantization, thereby contributing to more informed decisions in model optimization and deployment. Therefore, we present the accuracies of the original and compressed models to demonstrate the relative change in model accuracy.

3) Transform Quantization: Transform quantization decorrelates and quantizes the weights, which first rotates the space

and then quantizes the data after rotation with mixed precision as shown in Fig. 1(d). Because the redundancy between the dimension of weights can be removed after decorrelation, transform quantization could quantize weights to lower bit width in the transformed space. A unified framework was proposed [51] to enable low bit-rate quantization of deep neural networks by combining quantization and dimension reduction (decorrelation) techniques. The framework introduces a theory of rate and distortion for neural network quantization, which formulates optimum quantization as a rate-distortion optimization problem. Optimal End-to-end Learned Transform (ELT) was then derived to solve optimal bit width allocation following decorrelation. Experimental results demonstrated that transform quantization significantly improved the state-of-the-art in neural network quantization.

A similar idea was explored in other neural network compression works. Wang *et al.* [52] proposed a technique to compress neural networks that involved spatially decorrelating the weights using the DCT, followed by vector quantization on the decorrelated weights. Alongside the DCT approach, other methods aimed to reduce the dimension of neural network layers using principal component analysis (PCA) and related techniques. [53] employed singular value decomposition (SVD) to perform low-rank projection across kernels in convolution layers or across the rows and columns of weight matrices for fully-connected layers. Similarly, [54] extended this method using the generalized SVD. [55] further applied SVD on all four axes of convolutional tensors. Additionally, Li *et al.* [56] partitioned the kernels into subsets and performed SVD on each subset of kernels separately.

4) Quantization for Transformers: Transformers [57], [58] now stand as the leading architecture for a range of computer vision tasks, encompassing image classification, object detection, and semantic segmentation. However, achieving high-accuracy performance in transformers comes with the trade-off of substantial computational complexity, often involving tens of millions or even more parameters in a model. The substantial number of parameters poses a significant challenge for deploying transformers on mobile devices and hinders their practical applications. As a result, the quantization of transformers becomes more important and has received attentions.

[59] observed that activations of the transfer have high dynamic activation ranges and structured outliers and thus propose a per-embedding-group quantization scheme. [60] introduced a mixed-precision decomposition scheme, which isolates the outlier feature dimensions into a 16-bit matrix multiplication. Liu *et al.* [61] recently introduced post-training quantization for transformers, integrating ranking-aware loss, bias correction, and mixed-precision techniques. In particular, the inclusion of a ranking-aware loss aimed to preserve the relative order of quantized attention maps within transformers. Additionally, a bias correction method was implemented to adjust the distribution of quantized weights and activations. The exploration of mixed-precision quantization involved assigning more bit widths to layers with higher sensitivity, thus ensuring optimal performance retention. Ding *et al.* [62] went a step further in enhancing by refining the calibration process following linear layers.

5) Entropy Coding: Quantized values can be further compressed using entropy coding in a lossless manner. There are two types of entropy coding methods [63]: Fixed-to-Variable Length (FVL) entropy coding and Variable-to-Fixed Length (VFL) entropy coding. FVL coding maps a fixed number of symbols to variable-length codes, using arithmetic or Huffman coding. FVL entropy coding methods achieve high coding efficiency (high compression ratio), but their decoding process is inefficient as it decodes codes bit by bit, leading to slower processing. It has comparable coding efficiency as Huffman coding, with $10\times$ to $15\times$ faster decoding speed. On the other hand, VFL coding maps a variable number of symbols to fixed-length codes, enabling byte-by-byte decoding, resulting in much faster decoding compared to FVL coding. Tunstall coding [64] is a popular VFL coding method that achieves comparable coding efficiency as Huffman coding but offers decoding speeds that are $10\times$ to $15\times$ faster.

Several studies in the literature have utilized entropy coding to compress DNNs. One such approach is the Deep Compression framework [25], which includes pruning, quantization, and Huffman coding. Similarly, Coreset-Based [46] Compression coupled quantization with Huffman coding and exploited the weight redundancies to improve the compression ratio. As mentioned above, the decoding of Huffman coding is inefficient, potentially slowing down the inference stage. To address this, DeepCABAC [47] proposed a context-adaptive binary arithmetic coding to compress DNNs. EPR [48] represented the weights of neural networks in a latent space and used arithmetic coding to compress the representation. To address this, DeepCABAC and EPR employed arithmetic coding to obtain a higher compression ratio, although arithmetic coding has the most increased computational complexity, which is very difficult to implement. RDOC [49] adopted Tunstall coding to obtain superior compression capability and fast decoding speed. Table III summarizes the works that utilize entropy coding for neural network compression.

In summary, different quantization approaches have advantages and disadvantages regarding the computational complexity, quantization loss, hardware implementation, etc. In uniform quantization, the space is divided into equidistant quantization centroids. Such quantization scheme is straightforward for implementation without additional overhead involved, but the quantization loss is relatively high. Non-uniform quantization is able to achieve less quantization loss because the quantizer is more flexible and can divide the space into different clusters based on the distributions of data. The drawbacks of non-uniform quantization is that it requires multiple code-books to store the quantization centroids. Mixed-precision quantization uses different bit widths to quantize different parts of parameters. It is more reasonable than uniform quantization because different parts of parameters may react distinctively to quantization. However, specific hardware is needed to support the computation with mixed precision (*e.g.*, 4-bit vs. 8-bit). Different with all the others, transform quantization firstly rotates the space before quantization to remove redundancy between the dimensions. As a result, lower bit widths are able to be achieved after the space transform. The problem is that a large rotation matrix is usually needed

TABLE IV
CLASSIFICATION OF DIFFERENT PRUNING ALGORITHMS BY THEIR TYPE, STAGE OF DOING PRUNING, AND WHETHER THEY ARE STRUCTURED (DENOTED BY 'S') OR UNSTRUCTURED (DENOTED BY 'U'). MORE COMPREHENSIVE SUMMARY OF PRUNING AT INITIALIZATION SCHEME CAN BE FOUND IN [68].

Pruning methods	Pruning with training
Gradient based	Optimal Brain Damage [69] [U], WoodFisher [70] [U], MFAC [71] [U], HALP [72], Taylor [73], RD [74]
Customized criteria based	DPF [75] [S,U], DST [76] [U], Molchanov *et al.* [77] [U], SCOP [78] [S]
Regularization based	STR [79] [S,U], Savarese *et al.* [80] [U], Wang *et al.* [81] [S,U]
Magnitude based	RigL [82] [S,U], LAP [83] [S,U], GMP [84] [U], LAMP [85] [U], Global MP [86] [U], DSR [87] [U], SM [88] [U]
Reinforcement learning based	PuRL [89] [U], AMC [90] [S,U], RNP [91] [S]

for the space transform.

B. Network Pruning

Pruning is a widely used technique for compressing neural networks by removing neurons from networks. This helps reducing the storage size required for the network parameters, as well as the memory access during inference to load the surviving weights (neurons) from off-chip to on-chip memory. Over the years, numerous algorithms for pruning have been developed.

1) Pruning Strategies - Before, During, and After Training: Pruning solutions can be categorized according to multiple schemes. One scheme is when pruning is done with respect to the training of the model. Three choices exist here - pruning before training, during-training (pruning with training), or post-training (pruning after training).

Pruning before training It aims to identify sparse structures in dense network at initialization and directly train the sparse network, with the merit of accelerating training and back-propagation. SNIP [65] set the pruning criterion as the normalized magnitudes of gradients while GraSP [66] pruned those weights whose removal will result in least decrease in the gradient norm after pruning. FORCE [67] presented a modified saliency metric based on [65].

Pruning with training It starts with a dense network from scratch and gradually prunes out more connections from the networks during training. Researchers proposed different pruning criteria (*e.g.*, estimation of inverse Hessian in WoodFisher [70] and MFAC [71], Taylor-based criterion in Molchanov *et al.* [77], $L2$ regularization based pruning [81], magnitude-based criterion [82] and reinforcement learning based methods [89]), to remove less important parameters. Some approaches either finished the pruning in one training pass (*e.g.*, DPF [75]) or multiple training passes [92]. Another important technique is **prune-and-regrow** [82]. Specifically, instead of removing a neuron once and for all during a pruning cycle in the training, neurons are given chances to grow back later. RigL [82] droped parameters using parameter magnitudes and grow the connections with the highest magnitude gradients. DSR [87] randomly selected pruned neurons to grow back during training but keeps the active neurons in the whole network the same throughout. SM [88] pruned weights with small magnitude and growed new weights to fill in missing connections with the highest momentum magnitude. A comprehensive summary of the above pruning methods during training can be found in Tab. IV. In summary, pruning before or during training offers the advantage of reducing the computational requirements throughout training, specifically in terms of floating-point operations per second (FLOPs).

Post-training Pruning Different from pruning with training, post-training pruning typically conducts one-shot pruning after the model has been trained on a dataset [93]–[95]. Post-training pruning in the context of Transformers has drawn significant interest due to growing concerns about the substantial resources and time needed for re-training. [94] proposed a post-training compression framework which covers both weight pruning and quantization in a unified setting based on an efficient realization of the classical Optimal Brain Surgeon (OBS) framework [69]. [95] introduced a fast post-training framework that automatically prunes the Transformer model using structured sparsity methods.

2) Pruning Paradigms - Unstructured, Structured, and Semi-Structured Pruning: Another essential consideration is whether pruning occurs at the weight or channel level. It can be categorized into three types: unstructured pruning, structured pruning, and semi-structured pruning.

Unstructured Pruning Unstructured pruning refers to remove individual weights. Table II-B2 summarizes the top-performing algorithms for unstructured pruning. Among them, global magnitude pruning (Global MP) [86] achieved the top performance in terms of sparsifying parameters. This method ranks the weights by their absolute magnitude and prunes the smallest ones, resulting in high parameter sparsity. On the other hand, STR [79] achieved the top performance in terms of sparsifying FLOPs. It reparameterizes the weights using a soft threshold operator and learns the sparsity rates using gradient descent. GMP [84] developed a simple gradual pruning approach that requires minimal tuning. DNW [96] provided an effective mechanism for discovering sparse subnetworks of predefined architectures in a single training run.

TABLE V
RESULTS OF UNSTRUCTURED PRUNING ON RESNET-50 ON IMAGENET.

Methods	Top-1	Params	Sparsity	FLOPs pruned
ResNet-50	77.0%	25.6M	0.00%	0.0%
GMP [84]	75.60%	5.12M	80.00%	80.0%
DSR*# [87]	71.60%	5.12M	80.00%	69.9%
DNW [96]	76.00%	5.12M	80.00%	80.0%
SM [88]	74.90%	5.12M	80.00%	-
SM(ERK) [82]	75.20%	5.12M	80.00%	58.0%
RigL* [82]	74.60%	5.12M	80.00%	77.5%
RigL(ERK) [82]	75.10%	5.12M	80.00%	58.0%
DPF [75]	75.13%	5.12M	80.00%	80.0%
STR [79]	76.19%	5.22M	79.55%	81.3%
Global MP (One-shot) [86]	**76.84%**	5.12M	80.00%	72.4%
Global MP (Gradual) [86]	76.12%	5.12M	80.00%	76.7%

* The first layer of the network architecture is dense.
# The last layer of the network architecture is dense.

Structured Pruning Structured pruning targets the removal of organized neurons, such as channel-wise or block-wise pruning. Certain studies [97], [98] established metrics, ex-

cluding $l1$-norm [99], to decide specific channel pruning. HRank [98] mathematically proved that filters with lower-rank feature maps contain less informative content, suggesting their suitability for removal. Another research direction explores automatic pruning without meticulous hyper-parameter adjustments, with DHP [100] introduced a differentiable pruning technique using hypernetworks. This approach allows for automatic network pruning by using hypernetworks that take latent vectors as input and generate weight parameters for the backbone network. In parallel, a research line focuses on simultaneously pruning diverse redundant structures [101], [102]. For instance, GAL [101] achieved simultaneous end-to-end pruning of diverse structures, such as channels, filters, and blocks, using label-free generative adversarial learning. GAL employs a sparse soft mask to scale specific structures' outputs to zero. 3D [102] pruned a model by determining the optimal value of a pruned network along three dimensions, *i.e.*, layers, filters, and image resolution. It did this by conducting multiple pruning runs to build a database of pruning rates along each dimension and the associated accuracy. It then used the Lagrangian theorem to fit the data to an optimal pruning function. Table VI presents the top-performing algorithms for structured pruning.

TABLE VI
STRUCTURED PRUNING OUTCOMES FOR RESNET-50 ON IMAGENET.

Methods	Top-1 Acc	Params	Sparsity	FLOPs pruned
ResNet-50	76.15%	25.6M	0.00%	0.0%
GAL [101]	71.95%	21.25M	17.00%	43.0%
FPGM [97]	74.83%	-	-	54.0%
HRank [98]	74.98%	16.13M	37.00%	44.0%
DHP [100]	75.45%	11.78M	54.00%	49.9%
PScratch [99]	75.45%	**9.22M**	**64.00%**	**49.9%**
3D [102]	**75.90%**	12.03M	53.00%	**49.9%**

Semi-structured Pruning Aside from the two major sparsity patterns, semi-structured pruning scheme is also adopted by few literature, which aligns the sparsity patterns with the GEMM (GEneric Matrix Multiplication) designs to achieve handy hardware acceleration while still benefits from relatively low accuracy drop. [103] pruned feedforward weights in Transformer-based language models (*e.g.*, BERT) in block-sparsity pattern and achieved a trade-off between accuracy retaining and acceleration. Apart from the regular block shaped type of semi-structure, another N:M sparsity [104] arises recently available on NVIDIA's A-series GPUs. Several Vision Transformers pruning works [105]–[107] explored using such sparsity scheme to achieve fine-grained feed-forward and attention pruning.

By examining Table II-B2 and Table VI, it becomes evident that structured pruning is conducive to hardware constraints. Nevertheless, it prunes fewer neurons and FLOPs, resulting in lower memory costs when compared to fine-grained unstructured pruning while maintaining similar levels of accuracy. More memory analysis of pruning can be found in [108].

There are some interesting works on exploring network pruning to meet specific computational constraints. One important technique is **slimmable network** [109]–[111]. Instead of optimizing a fixed and unified sparse network for inference, slimmable network aims to dynamically determine the network widths (*e.g.*, layers, channels, or blocks) at runtime, to predict the sample with minimal performance sacrifice. [109] was one of the pioneer works in slimmable network, which proposed networks with variable width (number of channels) and an altered BatchNorm strategy to train the network with switchable widths by sharing network parameters. [111] adopted learnable gating layers to decide the layer widths. Another important research line is **Once-for-All** networks [112]. Instead of training multiple models for various resource constraints, Once-for-All networks learn a shared set of parameters that can be applied to different subnetworks. These subnetworks vary in depth, width, or other architectural aspects, allowing the model to adapt to diverse computational requirements.

In summary, network pruning has a long history and its effectiveness has been verified by numerous research works. Unstructured pruning is the de-facto scheme known for best performance preservation, but limited by the difficulties to deploy on real hardware to achieve speedups. The new N:M sparsity scheme on NVIDIA's Ampere platform offers a great fine-grained sparsity pattern similar unstructured pruning and with remarkable acceleration, but the restriction of row-wise sparsity constraint and the limited availability are still outstanding issues to address. Block-structured and Structured pruning look at larger structures in the networks as elements for pruning which is more hardware friendly. However, the dependency of neurons within a pruning group (*e.g.* A 2d block of neurons for block-structured or all the neurons in a channel in structured pruning) poses challenge to evaluate the impact of removing the whole pruning group. The optimal pruning criteria for minimizing accuracy loss while removing redundancy remains an open question, especially considering the continuous evolution of novel neural network architectures. Classic criteria such as magnitude-based and derivative-based importance are originally proposed under post-train-pruning, where a prune-then-finetune procedure is required. These criteria are further borrowed into more advanced schemes such as prune-at-initialization and prune-during-training, where pruning are interleaved with the training.

C. Knowledge Distillation

Knowledge Distillation (KD) compresses the learned knowledge from a large model (Teacher) into a smaller model (Student), which is designed to meet the latency and footprint requirements on resource-limited devices. This compression results in fewer trainable parameters and FLOPs while maintaining similar accuracy to the teacher model. The student architecture is flexible and can be optimized for hardware efficiency. Due to its flexibility and efficiency in model compression, it has drawn a lot of interest and has also been verified in many research areas such as computer vision [113]–[116], neural language processing [117]–[119], speech recognition [120]–[122], time series regression [123], [124] and so on. Two key challenges in KD research are defining what constitutes knowledge and developing effective methods for transferring that knowledge to the student model. In this subsection, we first provide an overview of the structured knowledge in the literature, before delving into a discussion of the popular distillation schemes.

1) Structured Knowledge: Generally, knowledge can be broadly categorized into three types: knowledge from logits, features, and relations.

Knowledge from logits The terminology of "knowledge distillation" was first introduced by Hinton *et al.* [125]. The logits from the softmax layer of a complex teacher model, softened by a temperature factor, were employed as the "dark knowledge" to guide the training process of a compact student model. Compared with ground truth labels, the softened logits serve as a regularization term to improve the generalization capability of the student model. Zhang *et al.* [126] formalized the logits from multiple teachers as a graph and presented a logit graph distillation approach for video classification. More recently, Zhao *et al.* [127] reformulated teacher's logits into two parts: TCKD and NCKD, which are a binary probability vector for target class and an independent probability vector for non-target classes, respectively. They empirically showed that the NCKD is critical to the success of distillation and therefore proposed to decouple them by introducing two independent hyper-parameters.

Many KD works have shown the effectiveness of using logits-based knowledge [128]–[130]. However, this type of knowledge is originally designed for classification tasks but not for regression tasks where the output is a scalar. Despite this, recent works such as [123], [124], [131] have demonstrated the effectiveness of logits-based knowledge on time series regression tasks. By directly minimizing the distance between the teacher and student's scalar predictions, the generalization capability of the compact student can be improved.

Knowledge from feature representations Utilizing the feature representations from the teacher's intermediate layers as the knowledge has received significant attention as they contain more meaningful information than logits. Directly aligning teacher and student's features via a specific distance metric can force a student to learn similar representations to its teacher. However, direct regression of teacher and student's feature maps can lead to suboptimal performance, resulting in less effective knowledge transfer. Therefore, other well-structured knowledge derived from feature representations was further presented, such as attention maps [132], the flow of solution procedure [133], factor transferring [134], mutual information [135] and more. Furthermore, Liu *et al.* [136] proposed using the spatial correlations between different channels of features as the knowledge to align the diversity and homology of feature space between the teacher and student. direct regression teacher and student's feature maps. Lin *et al.* [137] addressed the sub-optimal issue of direct regression of teacher and student's feature maps by reconstructing the student's feature maps with a novel target-aware transformer. Recently, Shang *et al.* [138] formulated the knowledge as Lipschitz constants via a transmitting matrix derived from the input and output feature maps of each network block. Huang *et al.* [139] argued that we should encourage the student not only to inherit the knowledge from the teacher but also to explore more diverse feature representations via designing a novel dissimilarity loss among feature representations. Finally, Ji *et al.* [140] explored the similarity relations between teacher and student's feature maps from different layers via an attention network to avoid manual layer pairing.

Knowledge from instance relation Instead of simply mimicking teachers' responses via logits or feature representations on individual training instances, the relationships between different instances could also serve as knowledge. For example, Passalis and Tefas [141] modeled teacher's knowledge as a probability distribution among a batch of samples and transferred it by minimizing the distribution divergence between teacher and student. In a similar vein, Park *et al.* [142] proposed transferring the mutual relations instead of actual representations by introducing two novel distillation losses, allowing student to learn the structured relations among different samples. Liu *et al.* [143] formulated the instance relationship graph as the knowledge source, and Huang *et al.* [144] aligned the inter-class and intra-class relation simultaneously for instances from different classes and same class, allowing for knowledge transfer from stronger teachers or more robust training strategies. Similarly, Yun *et al.* [145] minimized the intra-class variance among different instances having the same labels. Moreover, contrastive learning has been introduced for student representation learning and sample relation optimizing [124], [146], [147]. Specifically, the teacher and student-generated outputs from the same sample are generally considered positive pairs, while the outputs of teachers generated from different samples are regarded as negative pairs. Contrastive learning minimizes the distance of positive pairs and maximizes the distance of negative pairs.

To sum up, although the effectiveness of the aforementioned structured knowledge has been verified by many existing works, they have their advantages and disadvantages. The logit-based knowledge is the most widely-used knowledge due to its superior simplicity. However, the limited information provided by the logits severely hinders the efficacy of knowledge transformation. Meanwhile, its performance on other tasks (*e.g.*, regression task) still requires more exploration. On the contrary, feature representations can offer more informative knowledge than logits for classification and regression tasks. However, they have a drawback that task-specific knowledge from feature representations may not be useful for other tasks. When distilling knowledge between different network architectures, pre-defined feature-based knowledge can harm the performance of students [123]. Choosing the right intermediate layers for distillation is challenging, especially when the teacher and student models differ significantly in capacity. Combining logit-based and feature-based knowledge is a common strategy in many KD-related works. Lastly, although some works have demonstrated that relation-based knowledge distillation works well, it does involve more complex training processes [24], [148], like extra memory for data storage and graph construction. Generally, it is also very challenging to integrate with the other two types of structured knowledge.

2) Distillation Schemes: Distillation schemes pertain to efficiently transferring knowledge from the teacher model to the student model, categorized into three types briefly: a) metric-specified vs. metric-free distillation; b) teacher-specified vs. Teacher-free distillation; and c) data-available vs. data-free distillation. More details can be found in Appendix C.

D. Non-linear operations

Most existing works focus on linear matrix operations in convolutional layers or fully connected layers which account for over 99% of the total operations in modern DNNs [149]. The implementation of non-linear blocks, such as Softmax and Sigmoid, presents significant difficulties for hardware. This is due to two main reasons. Firstly, convolutional operations are much simpler and cheaper to perform using a multiplier and adder, whereas non-linear functions like exponential require a more complex hardware unit. Secondly, the compute ratio per unit area for non-linear blocks is currently an order of magnitude lower than that for convolutional layers in existing accelerators. However, there has been a recent increase in research efforts to optimize non-linear operations for hardware implementations. In this section, we review the literature which aims cost-efficient inference for non-linear operations.

1) NMS: The greedy non-maximum suppression (NMS) algorithm [150] (Algorithm in Appendix D) sorts bounding boxes by class confidence scores and removes boxes with significant overlap (IoU) iteratively.

Table VII provides an overview of various NMS algorithms. In particular, except greedy NMS, we summarize them into six types. The first three types intend to optimize the sub-module in the greedy algorithm, whereas the last three seek an entire algorithm replacement. The six types are: **1) greedy-sorting**: In the greedy NMS, the boxes are sorted based on the categorical confidence but not the location confidence, and thus the ranking list might not be reliable. This type of methods [151]–[153] focuses on producing a more reliable ranking list by incorporating extra location-based information into the ranking. For example, SoftNMS [151] decayed the confidence score of the remaining detection boxes using a function with the IoU as an argument, thereby affecting the sorting. **2) greedy-finalizing box**: it focuses on finalizing the box coordination [154], [155]. **3) greedy-duplicate check**: these methods focus on duplicate bounding box check criteria [156], [157]. For example, [156] dynamically modified the IoU threshold in the duplicate check using object density. **4) end-to-end**: these methods replace NMS with a neural network sub-module and train in an end-to-end manner without manual intervention [158]–[160]. **5) clustering**: this kind of methods focuses on parallel clustering predicted bounding boxes to speed up the NMS [161], [162]. **6) pooling**: it focuses on reformulating NMS as a max pooling operation which is inherently parallel and hardware-friendly [163]–[165].

2) Softmax: Softmax is a general operation to predict the class probability in many tasks. Its equation is $\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ where z_i denotes the i-th element of the input vector $\mathbf{z}$. K denotes the number of classes. Softmax involves expensive non-linear operations such as exponentials and divisions, and its computational cost can become prohibitively expensive when dealing with a large number of classes.

Researchers have developed methods to speed up softmax computation in Natural Language Processing (NLP), as it can dominate the complexity of neural networks with large vocabularies. Table VIII summarizes various softmax approximation methods in the literature, which can be categorized into five types: 1) **class-based hierarchical softmax** [166]: . A tree structure is constructed to estimate the softmax value along the tree depth, avoiding the need to traverse the whole vocabulary. **2) sampling-based softmax**: it chooses a small subset of possible outputs and trains only with those. **3) differentiated softmax**: this restricts the effective parameters, using the fraction of the total output matrix. The matrix allocates a higher dimensional representation to frequent words and only a lower dimensional vector to rare words. **4) self-normalization**: it employs an additional training loss term [174], which leads to a normalization factor close to 1. **5) softmax replacement**: a new probability computation function is proposed for accuracy improvement [171] or computational saving [169].

3) Activation functions: Activation functions like ReLU [175], PReLU [176], and Sigmoid are crucial in neural networks and should be nonlinear, differentiable, continuous, bounded, and zero-centered. The implementation of activation functions with linear operations is easy, while exponential or complex nonlinear operations may require linear approximations for faster inference. Choosing an appropriate activation function depends on the specific neural network requirements and the available computational resources.

To enhance the efficiency and performance of neural networks, the optimization of non-linear blocks, including activation functions like softmax and sigmoid, is a crucial focus of research. The challenges arise from the computational complexity of these functions, particularly in hardware implementations where simpler operations, such as those in convolutional layers, are more cost-effective. Recent efforts have been directed towards devising hardware-friendly activation functions, employing quantization techniques, utilizing look-up tables, and exploring fixed-point arithmetic. These optimization endeavors aim to strike a balance between computational efficiency and model accuracy, facilitating the deployment of neural networks in diverse hardware environments, including edge devices and real-time systems.

E. NAS and TinyML

1) NAS: The research on Neural Architecture Search (NAS) for efficient model design has yielded significant advancements in automating the creation of compact yet high-performing neural networks. Notable works such as EfficientNet [177], ProxylessNAS [178], MnasNet [179], FBNet [180], DARTS [181] and Once-for-All [112], showcase various approaches to NAS that consider efficiency in terms of model size, computational resources, and hardware constraints. These methods leverage techniques like gradient-based optimization [181], platform-awareness [180], parameter sharing [58], [182], various network operations [57], and specialized network training [112] to find architecture configurations that strike a balance between efficiency and performance.

2) TinyML: The study of Tiny Machine Learning (TinyML) has garnered significant attention due to its focus on deploying deep learning models on edge devices with limited resources, such as IoT devices. There is a growing body of literature and available resources in this field. These methods make use of various techniques, taking into account hardware constraints,

TABLE VII
COMPARING NMS ALGORITHMS IN THE LITERATURE

Methods	Types	Network Replacement	Parallelable?	Two-stage detector?	Time Complexity
GreedyNMS [150]	greedy	✗	✗	✓	$\mathcal{O}(n\log(n))+\mathcal{O}(nm)$
SoftNMS [151]	1)	✗	✗	✓	$\mathcal{O}(n\log(n))+\mathcal{O}(nm)$
AdaptiveNMS [156]	3)	✗	✗	✓	$\mathcal{O}(n\log(n))+\mathcal{O}(nm)$
TNetNMS [158]	4)	✓	✓	✓	Nework architecture related
FeatureNMS [157]	3)	✗	✗	✓	$\mathcal{O}(nm)$
vg-NMS [154]	2)	✗	✗	✓	$\mathcal{O}(n\log(n))+\mathcal{O}(nm)$
MaxpoolNMS [164]	6)	✗	✓	✗	$\mathcal{O}(n)$
PSRR-MaxpoolNMS [165]	6)	✗	✓	✓	$\mathcal{O}(n)$
ClusteringNMS [161]	5)	✗	✓	✗	$\mathcal{O}(nm)$
GPUNMS [162]	5)	✗	✓	✗	$\mathcal{O}(nm)$
FitnessNMS [152]	1)	✗	✗	✓	$\mathcal{O}(n\log(n))+\mathcal{O}(nm)$
SofterNMS [155]	2)	✗	✗	✓	$\mathcal{O}(n\log(n))+\mathcal{O}(nm)$
GnetNMS [159]	4)	✓	✓	✓	Network architecture related
IoU-Net [153]	1)	✗	✓	✓	$\mathcal{O}(n\log(n))+\mathcal{O}(nm)$
RelationNetNMS [160]	4)	✓	✓	✓	Nework architecture related

TABLE VIII
COMPARING SOFTMAX ALGORITHMS IN THE LITERATURE

Methods	Types	Stage
GPUSoftmax [166]	1)	train
SvdSoftmax [167]	3)	inference
AdaptiveSoftmax [168]	2)	train
SOFT [169]	5)	train & inference
ANNSoftmax [170]	2)	train
Sigsoftmax [171]	5)	train & inference
L2S [171]	1)	inference
DS [172]	1)	inference
RF-Softmax [173]	2)	train
RegularizerSoftmax [174]	4)	inference

including factors like energy consumption, Random Access Memory (RAM), and multiply–accumulate (MAC) operations. For example, Ancilotto *et al.* discussed considerations related to hardware constraints in their work [183]. Additionally, Lin *et al.* addressed the issue of imbalanced memory distributions in Convolutional Neural Networks (CNNs) by implementing patch-by-patch scheduling [184]. Cai *et al.* proposed a method to reduce memory usage during training by freezing weights in transfer learning [185].

III. WHEN NN COMPRESSION MEETS HARDWARE

A. Overview

Performance, power, and area (PPA) are the three most important metrics in evaluating hardware cost. Table II in Appendix E summarizes the quantitative and relative energy and area consumption of various operations such as adding and multiplication in different data formats. We observe that energy efficiency can be achieved through good data addressing, as data reading is relatively expensive. It costs 166.7× and 1,666.7× more energy to read 32-bit data from an 8KB static random-access memory (SRAM) and 1MB SRAM than perform an 8-bit integer addition, respectively. In addition, quantizing float-point data to integers, from high precision (*e.g.*, 32-bit) to low precision (*e.g.*, 8-bit), reduces energy consumption. For instance, a 32-bit float-point addition requires 30× more energy than an 8-bit integer addition. Finally, the trend of the area costs of different operations is similar to their energy costs.

For the linear operations in DNNs, quantization, and pruning *etc*. can be applied to reduce the number of weights or even activations, hence optimizing the PPA of the hardware. For the non-linear operations, however, hardware-friendly approximation algorithms are needed to save the hardware cost.

Hardware accelerators generally target either Field Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs). One accelerator can either be mapped into the FPGA flexibly for quick validation and further design space explorations or be fabricated as a high-performance ASIC chip when its functionalities and architecture are frozen and it is well-validated.

B. Hardware Accelerating on Linear Operations

1) Hardware Accelerating on CNNs and Transformers: With the rapid advancements of DNNs, the corresponding hardware accelerators have become increasingly important. Most CNN accelerators use direct convolution as their primary computation method, which accumulates the product of the inputs and corresponding weights in a certain dataflow. Members of DianNao family [186]–[188] computes convolutions directly. DianNao [186] exploited the locality properties of large-scale layers, achieving impressive results with only 3 mm^2 at 65 nm. However, the memory wall becomes the bottleneck when computing classifier and convolutional layers with private kernels [187]. To address this, DaDianNao [187] processed convolution with the data from nearby SRAM buffers and embedded dynamic random access memory (eDRAM) banks. NVDLA [189], an open-source hardware inference accelerator, provided direct convolution mode. A wide multiply-accumulate (MAC) pipeline was implemented

to parallel process the convolutional operations with memory bandwidth optimization. Another accelerator, Maeri [190] supported arbitrary dataflows by utilizing fine-grained reconfigurable tree-based interconnection network typologies to shape different sizes and numbers of virtual neurons.

By transforming the regular convolution into Generalized Matrix-matrix Multiplication (GEMM) or Generalized Matrix-vector Multiplication (GEMV), some matrix multiplication optimization algorithms, such as Strassen algorithm [191], Winograd algorithm [192] and Fast Fourier Transform (FFT) [193] can be applied to speed up the computation. For example, Cong *et al*. [194] applied the Strassen algorithm to the convolution transformed in GEMM and reduced the number of multiplications from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^{2.807})$. Lavin [195] and NVDLA [189] applied Winograd convolution to reduce the number of multiplications while increasing the adders for the transformation. The FFT algorithm converts the data to the more computationally efficient Fourier domain. Previous work [196] explored the application of FFT to accelerate inference, while more recent works have attempted to map FFT on GPU platforms [197], [198]. The architecture proposed by Liang *et al*. [199] also supported the FFT algorithm.

Over the past few years, the Transformer model has replaced Recurrent Neural Networks (RNNs) and CNNs in the Natural Language Processing (NLP) and Computer Vision (CV) domains. However, Transformer models have different computation patterns than traditional neural networks, with more weights and computations, necessitating careful hardware accelerator design. The hardware accelerating solutions for Transformers are reported in [200]–[206]. The accelerator for the multi-head attention (MHA) ResBlock and the position-wise feed-forward network (FFN) ResBlock in the NLP Transformer was first reported by Lu *et al*. [200]. In A^3 [201], an approximate method was applied for efficient attention computation. Vis-TOP [202] focused on accelerating Visual Transformers (ViTs) and presented a customized hardware architecture for a three-layer, two-level basic Transformer. VAQF [203] supported quantized ViTs with binary weights and low-bit activations on FPGA for inference. SwiftTron [205] directly mapped the ViT on ASIC with an area of 273 mm^2 and power of 33.64 W, adopting approximated non-linear functions proposed by NN-LUT [204], where they were computed via Piece-wise Linear Functions. The accelerator introduced by Nag *et al*. [206] targeted FPGAs with dedicated computation blocks for the GEMMs in the MHA.

In summary, hardware accelerators for DNNs can use direct convolution or optimization algorithms like Strassen or Winograd to accelerate computation. Furthermore, specialized accelerators have been created for Transformer models.

2) Accelerating Networks after Quantization: Mixed-Precision and Entropy Coding: The bit width of weights in different layers or channels can be different to balance model precision and size. Table IX summarizes the popular hardware accelerators that support multiple precision data formats. In [207], 8-bit and 16-bit fixed-point NN layers were supported. Zhang *et al*. [208] proposed a new flexible unit that supports five different precision in training and inference. LNPU [209] was a highly energy-efficient DNN accelerator that supports training and inference, achieving up to 25.3 TFLOPS/W energy efficiency when processing a highly sparse weight layer in 8-bit float-point data format. Zhou *et al*. [210] proposed a CNN accelerator that supports mixed precision computations both within-layer and layer-wise. By introducing a fine-grained mixed precision unit, the hardware enables within-layer mixed-precision operations, which reduces computation area by nearly 50% and dynamic power by around 12.1% in AlexNet and VGG16 compared to the baseline. Some works exploit the low-bit precision such as 4-bit or sub-4-bit. The accelerator proposed by Fleischer *et al*. [211] supported very low-precision data formats such as binary and ternary for aggressive inference performance. In addition, the 16-bit float-point data format is supported for high accuracy in training, hence achieving 24 Tera Operations Per Second (TOPS) inference performance in a 9 mm^2 chip. Wang *et al*. [212] introduced a flexible design flow that supports mixed-precision computations of hybrid neural networks with quantized activation and quantized weights targeting FPGAs. With parameterized computation engines, the introduced AccELB performs binary and ternary convolution without multiplications. The flexible 4-core AI chip introduced by Agrawal *et al*. [213] supported both training and inference with multiple precisions, *i.e*., INT2, INT4, hybrid-FP8, FP16, and FP32 with a performance of up to 102.4 TOPS.

Besides quantization, entropy coding can lower hardware costs while maintaining accuracy by encoding weights in a more condensed representation, resulting in fewer bits per variable [214]. This is accomplished by taking advantage of the peaked distribution of quantized values.

The coding schemes are classified into Fixed-to-Variable (F2V) and Variable-to-Fixed (V2F) entropy coding methods. F2V coding methods, such as arithmetic coding [215], [216] and Huffman coding [217], [218], encode a fixed number of symbols into variable-length codewords. Hence, it is challenging for the F2V coding methods to decode parallelly. F2V coding methods have very high computational complexity for decoding. The decoding complexity of F2V coding methods is $O(n \cdot k)$, where n is the number of codewords, and k is the reciprocal compression ratio. Conversely, V2F coding methods, such as Tunstall coding [219], encode multiple symbols to a fixed number of bits. During the decoding stage, it is feasible to process multiple bits simultaneously and decode multiple symbols per clock cycle. Additionally, parallel decoding of the encoded string is possible by dividing it into bit chunks of fixed length according to the codeword length.

Huffman coding algorithm first lists symbols in decreasing order of their probabilities. Then, it constructs a Huffman tree by adding two symbols with the smallest probabilities at each step and removing them from the list. Finally, an auxiliary symbol is added to represent the two removed original symbols. Arithmetic coding assigns one code to the entire input instead of coding individual symbols like Huffman coding. It starts with a specified interval and symbolically reads the input while narrowing down the gap.

In Deep Compression [218] and Coreset-base Compression [217], Huffman coding [220] was used to compress the quantized weights. However, in [215] and [216], another

TABLE IX
A SUMMARY OF MIXED-PRECISION HARDWARE AND THEIR COMPUTE DENSITY (TOPS/W OR TOPS/MM^2).

Methods	Supporting Format	Platform	Area	Technology	Inference Performance	Frequency	Training or Inference	Compute Density (TOPS/W or TOPS/mm^2)
Yin *et al.* [207]	INT8, INT16	ASIC	19.36 mm^2	65 nm lp	368.4 GPOS	200 MHz	Inference	1.28 TOPS/W
Zhang *et al.* [208] [a]	INT4	ASIC	2943 μm^2	28 nm	29.6 GOPS / unit	3.7 GHz	Both	4.81 TOPS/W
	INT8				14.8 GOPS / unit			2.09 TOPS/W
	INT16				7.4 GOPS / unit			1.01 TOPS/W
	FP8				14.8 GFLOPS / unit			1.10 TFLOPS/W
	FP16				7.4 GFLOPS / unit			0.55 TFLOPS/W
LNPU [209]	FP8	ASIC	16 mm^2	65 nm	600 GFLOPS	200 MHz	Both	3.48 TFLOPS/W (FP8, 0% sparsity)
	FP16				300 GFOPS			25.3 TFLOPS/W (FP8, 90% sparsity)
Zhou *et al.* [210]	INT8, INT16	FPGA	–	–	–	–	Inference	–
Fleischer *et al.* [211] [b]	Binary	ASIC	9 mm^2	14 nm	24 TOPS	1.5 GHz	Both	2.67 TOPS/mm^2
	Ternary				12 TOPS			1.33 TOPS/mm^2
	FP16				1.5 TFLOPS			0.17 TOPS/mm^2
AccELB [212]	Hybrid-INT1, INT2, INT4, INT8	FPGA	–	–	0.49-10.3 TOPS	200 MHz	Inference	–
Agrawal *et al.* [213]	INT2	ASIC	16 mm^2	7 nm	–	1.0-1.6 GHz	Both	–
	INT4				64-102.4 TOPS			8.9-16.5 TOPS/W
	Hybrid-FP8				16-25.6 TFLOPS			1.9-3.5 TFLOPS/W
	FP16				8-12.8 TFLOPS			0.98-1.8 TFLOPS/W
	FP32				–			–

[a] This work proposes one computation unit. Hence the frequency is computed by $1/$delay, where the delay is reported. The inference performance is computed by $2 \times$ #parallel op $\times$ frequency, where the number of parallel operations in one unit is reported. The energy efficiency is computed by $1/$energy #op, where the energy per operation is reported.
[b] The energy efficiency is not reported; area efficiency is reported instead.

TABLE X
THE HARDWARE IMPLEMENTATIONS OF VARIOUS CODING METHODS AND THE SIZE OF THE RESNET-50 AFTER COMPRESSION.

Methods	Coding Method	Coding Schemes	Parallelable	Memory Aligned	Size After Compression (MB)
Deep Compression [218]	Huffman	F2V	✗	✗	6.06
Coreset-base Compression [217]	Huffman	F2V	✗	✗	6.46
Oktay *et al.* [215]	Arithmetic	F2V	✗	✗	5.49
Deepcabac [216]	Arithmetic	F2V	✗	✗	6.06
Chen *et al.* [219]	Tunstall	V2F	✓	✓	5.85

TABLE XI
STORAGE OVERHEAD AND DECODING TAXONOMY FOR COMMON ENCODING METHODS. VECTOR d STORES n DIMENSIONS OF A TENSOR THAT CONTAINS NNZ NON-ZERO ELEMENTS [227].

Format	Storage Overhead (bits)	Decoding Taxonomy
COO	$NNZ \times \sum_1^n \lceil \log_2 d_i \rceil$	Direct
COO-1D	$NNZ \times \lceil \log_2 \prod_1^n d_i \rceil$	Direct
RLC	$NNZ \times B$	Single-step
Bitmap	$\prod_1^n d_i$	Single-step
CSR	$NNZ \times \lceil \log_2 d_1 \rceil + (d_0 + 1) \times \lceil \log_2 NNZ + 1 \rceil$	Double-step
CSC	$NNZ \times \lceil \log_2 d_0 \rceil + (d_1 + 1) \times \lceil \log_2 NNZ + 1 \rceil$	Double-step

Fixed-to-Variable (F2V) [221] coding method, arithmetic coding [222], was used. Nevertheless, it is challenging to develop parallel implementations for decoding, as F2V codewords are of variable length and cannot be indexed efficiently, making in-efficient indexing and preventing the decoding of multiple symbols per clock cycle. In [219], the Tunstall coding method was to compress ResNet and MobileNet after compression, where two Tunstall hardware decoders are designed and compared. Table X summarizes these coding methods. Table III in Appendix E [219] reports the 256 entries Huffman hardware decoder consumes $3.14\times$ more hardware resources and costs $6.23\times$ more time to decode the same amount of weights.

In summary, mixed-precision and entropy coding are adopted by hardware accelerators after the quantization of the DNNs for higher compute density and performance. However, aggressive quantization would introduce significant accuracy loss. Instead, entropy coding maintains accuracy while compressing the model. However, F2V coding methods challenge the parallel decoding on the hardware, which introduces extra overhead to the throughput, while the decoding process of the V2F coding methods, such as the Tunstall coding, is hardware-friendly and preferred.

3) Accelerating Networks after Pruning: Sparse Architecture: To reduce the energy efficiency significantly and boost computation speed with fewer computation units, recently works [209], [223]–[226] proposed accelerators that support sparse neural networks. As summarized in [227], there are five hardware accelerating methods for sparse neural networks:

- Storing the compressed data in off-chip memory to optimize the memory capacity requirement and improve energy efficiency;
- Storing the compressed data in on-chip memory to optimize the memory capacity requirement and improve energy efficiency;
- Skipping zero elements to improve energy efficiency;
- Reducing ineffectual computation cycles to improve performance and energy efficiency;
- Balancing the workloads of different processing elements to increase performance;

Lee *et al.* [209] proposed an architecture called LNPU that supports sparse DNNs with fine-grained mixed precision of FP8-FP16. Sparsity is exploited with intra- and inter-channel accumulation with the input load balancer (ILB) to alleviate the imbalanced workload problem caused by irregular sparsity, improving PE utilization. Only non-zero weights are kept in the internal buffers in Cambricon-X [228], while SCNN [225] compressed both non-zero weights and activations in both dynamic random-access memory (DRAM) and internal buffers.

Sparse encoding methods can reduce memory access, increase energy efficiency, and accelerate computation time for accelerators by adapting them to sparse tensors. This is because most sparsity encoding methods only store non-zero elements. These encoding methods are Coordinate (COO), COO-1D, Run-length Coding (RLC), Bitmap, Compressed Sparse Row (CSR), Compressed Sparse Column (CSC) [229] and Compressed Sparse Fiber (CSF) [230]. Table XI summarizes the storage overhead and decoding taxonomy for standard encoding methods suitable for sparse data [227] while Table XII summarizes the area and energy efficiency of the hardware accelerators that use sparsity encoding methods.

TABLE XII
THE HARDWARE ACCELERATORS THAT USE SPARSITY ENCODING METHODS

Methods	Technology	Area	Frequency	Energy Efficiency (TOPS/W)	Zero Elements Ratio	Encoding Format	Data Format
Eyeriss v2 [226]	65 nm	2695k gates (NAND-2)	200 MHz	0.96	68.98%	CSC	16b
Envision [240]	28 nm	1.87 mm^2	200 MHz	1.3	5-82%	Bitmap	16b
Sticker [232]	65 nm	7.8 mm^2	200 MHz	62.1	both 5%	COO for activation, CSC for weight	8b
SNAP [233]	16 nm	2.4 mm^2	260 MHz	21.55	both 10%	COO-1D	16b

The COO sparsity encoding method was adopted in [231], [232]. COO-1D was utilized in [233]–[235]. RLC was used in [209], [223], [225], [236]. Bitmap was used in [189], [232], [236], [237]. Mishra *et al*. [238] introduced CSR into the accelerator. Data in [224], [226], [238], [239] was compressed in CSC format. Extensor [231] utilized CSF encoding format.

More specifically, Eyeriss v2 [226] encoded the weights in CSC data format for both on- and off-chip accessing to reduce the bandwidth requirements and energy consumption. Hence $1.2\times$ and $1.3\times$ improvement in speed and energy consumption are obtained by introducing CSC into Eyeriss v2. To best utilize the benefit of CSC data format, the PE in Eyeriss v2 is specially designed, where there are registers to store both address vectors and data vectors in the CSC compressed data, which is drawn in Fig. 1 within Appendix E.

Sticker [232] compressed the sparse weight into CSC format offline and decodes weight on-chip. In addition, there are COO encoders and decoders in Sticker to adopt the COO format for activations online. Zhang *et al*. [233] designed an associative index matching (AIM) module in their hardware accelerator SNAP before ending the data into a multiplier array to find the non-zero partial sum position, *i.e.*, both the input activation and the corresponding weight are non-zero. A similar module was integrated in Cambricon-S [237] where the bitmap encoding method was deployed; hence the comparators in SNAP are replaced by AND gates, depicted in Fig. 2 within Appendix E. Envision [240] exploited sparsity by representing the data in bitmap format, storing those flags in a GRD memory, hence guarding both memory fetches and MAC operations. SCNN [225] employed a run-length encoding scheme, encoding the number of zeros between elements into the index vector, thus processing only non-zero weights and activations.

In summary, some works store the pruned weights in the memory to reduce the memory footprint and bandwidth requirements. Sparse encoding methods are introduced in the hardware accelerators for better performance and higher efficiency while introducing additional computational complexity to the decoding process.

C. Hardware Accelerating on Non-linear Operations

1) NMS: Although there are lots of algorithms that optimize NMS, hardware-accelerated NMS methods are explored less. The CPU-NMS [161] and GPU-NMS [162], [241] techniques are designed to operate on CPU and GPU platforms. GPU-NMS V2 [162] can process 1027 bounding boxes in just 0.324 ms on a GeForce GTX 1060 running at 1.70 GHz. Shi *et al.* [242] were motivated by GPU-NMS [241] to develop a power-efficient NMS accelerator that merges 1000 bounding boxes in 12.79 microseconds at 400 MHz. Despite this, the accelerated NMS still takes considerable time compared to the execution time of convolution operations. Moreover, these solutions lack support for high-resolution images.

MaxpoolNMS [164] and PSRR-MaxpoolNMS [165] reformulate NMS as MaxPool operation, which is inherently parallel and friendly to the hardware. Inspired by these two NMS algorithms, ShapoolNMS [163] is proposed, which is a salable NMS hardware accelerator that supports high image resolution and both one- and two-stage object detectors. These works are summarized in Table XIII. As the number of boxes increases, ShapoolNMS [163] surpasses GreedyNMS hardware, GPU-NMS V2 [162], and Shi *et al*.'s [242] solutions in terms of performance. This is because the time complexity of GPU-NMS and Shi *et al*.'s method is at least $\mathcal{O}(nm)$ [242], whereas ShapoolNMS has a time complexity of $\mathcal{O}(n)$, as outlined at Table IV in Appendix E.

The structure of ShapoolNMS [163] is depicted in Fig. 3 within Appendix E. According to [163], ShapoolNMS outperformed existing state-of-the-arts by an $8.54\times$ speedup and a $42,713\times$ speedup compared to GreedyNMS.

2) Softmax: Softmax involves expensive operations such as exponentiation and division, which is challenging for hardware platforms dealing with large input vectors. The exponential operations' wide range can also cause overflow problems with limited hardware resources [243]. Unfortunately, the Softmax layer can be a bottleneck for real-time applications as other convolutional layers can be hardware accelerated.

As summarized in Table XIV, the hardware acceleration of the softmax algorithm can be classified into three categories:

A. **Direct optimization**: optimize the hardware implementation of exponential and division units directly.
B. **Mathematical transformation**: Apply mathematical transforming (*e.g.*, logarithmic transforming) to replace the exponential operations.
C. **Mathematical reformulation**: Reformulate the de-facto softmax equation into other hardware-friendly equations, such as replacing the exponent base e to 2, new softmax layer based on integral stochastic computing.

Yuan [243], for the first time, proposed an efficient hardware implementation of softmax that uses a logarithmic transformation to eliminate division operations. Low-complexity subtractors replace complex division units. To address overflow issues, down-scale parameters are introduced in exponential units, reducing data width. The e^{x_i} operations are approximated by creating a lookup table (LUT) that stores the discrete results with the subset of x_i. The logarithmic unit is also based on LUT implementation. As the final results in [243] were based on a transformed version of the softmax equation, Du *et al*. [248] proposed a corrected version that the exponential function is applied to the final result. The exponential function is split into multiple basics based on the calculation rule. Hence a group of LUTs (GLUT) was introduced to store those

TABLE XIII
COMPARING THE NMS ALGORITHMS AND HARDWARE IMPLEMENTATIONS

NMS Algorithm/HW	Platform	Hardware Friendly?	High Resolution?	Parallelable?	Two-stage detectors?	Time Complexity
GreedyNMS [150]	CPU	✗	✓	✗	✓	$\mathcal{O}(n\log(n)) + \mathcal{O}(nm)$
SoftNMS [151]	CPU	✗	✓	✗	✓	$\mathcal{O}(n\log(n)) + \mathcal{O}(nm)$
MaxpoolNMS [164]	CPU	✓	✓	✓	✗	$\mathcal{O}(n)$
PSRR-MaxpoolNMS [165]	CPU	✓	✓	✓	✓	$\mathcal{O}(n)$
CPU-NMS [161]	CPU	–	✗	✓	✗	$\mathcal{O}(nm)$
GPU-NMS [162]	GPU	–	✗	✓	✗	$\mathcal{O}(nm)$
Shi *et al*. [242]	ASIC	–	✗	✓	✗	$\mathcal{O}(nm)$
ShapoolNMS [163]	ASIC	–	✓	✓	✓	$\mathcal{O}(n)$

TABLE XIV
THE SUMMARY OF SOFTMAX HARDWARE IMPLEMENTATIONS, OPTIMIZATIONS, AND AVERAGE MEAN SQUARED ERROR (MSE) OR ACCURACY LOSS.

Methods	Softmax Equation	Exponential Function Optimization Category[a]	Division Optimization Category[a]	Avg MSE	Accuracy Loss
Yuan *et al*. [243]	$\ln(\sigma(z_i)) = (z_i - (z)_{\max}) - \ln(\sum_{L=1}^{K} e^{z_j - (z)_{\max}})$	A (LUT-EXP)	B (logarithmic)	–	–
Geng *et al*. [149]	$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$	A (LUT-EXP, LUT-PLF)	A (bit shift)	–	–
Li *et al*. [244]	–	A (LUT-EXP)	A (optimized divider)	1.5×10^{-5}	–
Wang *et al*. [245]	$\sigma(z)_j = \exp(x_i - ln(\sum_{j=1}^{N} e^{x_j}))$ that $e^{y_i} = 2^{y_i \cdot \log_2 e}$	C (base 2)	B (exp-log)	8×10^{-5}	–
Sun *et al*. [246]	$\sigma(z)_j = e^{z_j - \max(z)} / \sum_{k=1}^{K} e^{z_k - \max(z)}$	A (GLUT-EXP)	A (bit shift)	10^{-8}	–
Hu *et al*. [247]	$\sigma(z)_j = \exp(-h'_j) exp(-ln \sum_{j=1}^{m} e^{-h_j})$	C (Integral SC)	B (logarithmic)	0.03	–
Du *et al*. [248]	$\sigma(z)_j = \exp((z_i - (z)_{\max}) - \ln(\sum_{L=1}^{K} e^{z_j - (z)_{\max}}))$	A (GLUT-EXP)	B (logarithmic)	–	0.24%
Kouretas *et al*. [249]	$\sigma(z)_j = e^{z_j - (z)_{\max}}$	A (LUT-EXP)	B (logarithmic)	–	0%
Wang *et al*. [250]	$\sigma(z)_j = \frac{N^{z_j}}{\sum_{i=1}^{j} N^{x_i}}$	C (dynamic learned)	–	–	–
Cardarilli *et al*. [251]	$\sigma(z)_j = \frac{2^{z_j}}{\sum_{k=1}^{N} 2^{x_k}}$	C (base 2)	C (approximated)	10^{-4}	–
Spagnolo *et al*. [252]	NA	A (accumulated)	A (bit shift)	–	0%
Stevens *et al*. [253]	NA	C (base 2)	–	–	0.5%

[a] The optimization categories are (A) Direct optimization, (B) Mathematical transformation, and (C) Mathematical reformulation. Detailed optimization methods are explained in brackets.

basics. Sun *et al*. [246] also proposed a similar architecture that utilizes GLUT to optimize the exponential function.

In addition to approximating e^{x_i} directly based on LUT (LUT-EXP), Geng *et al*. [149] also proposed another approximation technique using a Piecewise Linear Function (LUT-PLF). Besides, Geng *et al*. proposed six softmax operation combinations, including exponential functions, power-of-twos, and look-up tables. In [251], [245] and [253], the exponential function was optimized by replacing base exponent with 2. Hu *et al*. [247] reformulated the softmax layer using integral stochastic computing for the exponent operations and logarithmic transformation to avoid the division operations. Spagnolo [252] proposed approach optimizes the softmax through aggressive optimization, replacing complex processes with hardware-friendly operations. Specifically, right-shifting operations replace division, and addition operations replace exponential operations.

As Table XV summarized, most Softmax hardware accelerators target ASICs, with a relatively small area and power overhead compared with the convolution-based or Transformer-based hardware accelerators. However, none of them optimize the Softmax with a long input sequence or a lot of classes, crucial for Transformers. This gap presents a key opportunity to innovate Softmax accelerators for Transformer models.

D. Stochastic Computing Architecture

Power-efficient approximate computing techniques, Stochastic Computing (SC), is a random, non-weighted, bitstream-based unary computing technique. An SC circuit necessitates minimal hardware, low power consumption, and high fault tolerance to computational errors [254]. It has found widespread application in various network architectures, including CNNs, RNNs, and MLPs [255]. However, due to lengthy sequences and a high demand for stochastic number generators (SNGs), achieving competitive computational latency and energy consumption is challenging for SC neural networks. Recent advancements in stochastic computing techniques have greatly enhanced the performance of SC Neural Networks (SC NNs), bringing them to a level of competitiveness with conventional binary designs while using significantly fewer hardware resources. Some designs focus on improve the mostly cost random number generators (RNGs), leading to better hardware and energy efficiency [256]. Some research focus on designing efficient hardware architecture to implement SC NNs [257], [258]. [258] proposed a hardware implementation on stochastic computing for Radial Basis Function (RBF) neural networks.

IV. WHEN NN COMPRESSION MEETS HOMOMORPHIC ENCRYPTION

A. Motivation

Homomorphic Encryption (HE) stands as a distinctive encryption paradigm enabling computations on encrypted data (ciphertexts) without the requirement of the decryption key. The decrypted results of these computations are equivalent to outcomes derived from standard arithmetic operations performed on unencrypted data. Homomorphic Encryption can be categorized into Partial Homomorphic Encryption (PHE) and Fully Homomorphic Encryption (FHE) based on the supported homomorphic arithmetic operations. PHE supports either homomorphic addition, as introduced by Paillier [259],

TABLE XV
THE SUMMARY OF SOFTMAX HARDWARE ACCELERATORS.

Methods	Platform	Technology	Frequency	Area (LUT)	Power	Performance
Yuan *et al*. [243][a]	ASIC	90 nm	250 MHz	4.3576 $\times 10^4 \mu m^2$	3228.2 mW	10 classes
Geng *et al*. [149]	ASIC	65 nm	500 MHz	38 μm^2	0.086 mW	R-FCN (ResNet-101) on COCO 2014
Li *et al*. [244]	ASIC	45 nm	3.3 GHz	3.4348 $\times 10^4 \mu m^2$	–	16-bit input
Wang *et al*. [245]	ASIC	28 nm	2.8 GHz	1.5 $\times 10^4 \mu m^2$	51.60 mW	22.4 G/s (16-bit)
Sun *et al*. [246]	ASIC	65 nm	1 GHz	0.44 mm μm^2	333 mW	16-bit valid input
Hu *et al*. [247]	FPGA	–	–	10746 LUTs	603 mW	–
Du *et al*. [248]	ASIC	65 nm	500 MHz	0.64 mm^2	0.82 mW	Target on binary and ternary NN
Kouretas *et al*. [249]	ASIC	90 nm	255 MHz	2.5597 $\times 10^4 \mu m^2$	1576.3 mW	5 classes
Cardarilli *et al*. [251]	ASIC	90 nm	310 MHz	4.6816 $\times 10^4 \mu m^2$	2.769 mW	8-bit integer, 10 classes
Spagnolo *et al*. [252]	ASIC	28 nm	2.5 GHz	3595 μm^2	NA	16-bit integer, 10 classes

[a] The data is reported in [249].

or homomorphic multiplication, as outlined by Rivest [260], between two ciphertexts. In contrast, FHE, a more comprehensive variant pioneered by Gentry [261], accommodates both homomorphic addition and multiplication on ciphertexts. Additionally, FHE incorporates the *bootstrapping* primitive, although its practical performance remains a challenge, it holds the potential to reduce noise in ciphertexts, rendering them amenable for arbitrary depth of computations.

Given the powerful features of FHE, it is a natural choice of technology for protecting data privacy during computation in untrusted environment. Notably, the concept of Homomorphic Neural Networks (HNNs) involves the utilization of FHE to encrypt both neural network models and input data. This application ensures the confidentiality of models and data throughout training or inference processes conducted in untrusted settings, such as cloud computing or multi-party collaborations. This becomes particularly crucial in domains where data privacy holds paramount importance, as evidenced in sectors like medical or financial applications [262].

Fig. 2 illustrates the application of Homomorphic Neural Networks (HNN) in preserving the privacy of medical images for disease diagnosis within a cloud deployment, which is also referred to as the Machine Learning as a Service scenario. The process begins with the client generating HE keys to encrypt her private images, transforming them into ciphertexts denoted as $Enc(X)$. Subsequently, these ciphertexts are transmitted to the cloud server to request disease diagnosis on the encrypted images. The cloud server then performs evaluations using the HNN-based disease diagnosis model on the image ciphertexts, yielding the encrypted diagnosis result, denoted as $Enc(Y)$. This encrypted result is relayed back to the client, who decrypts it into plaintext using her secret decryption key. Notably, throughout this entire process, the cloud server remains incapable of accessing any input data from the client or intermediate data during HNN evaluation in plaintext, as it lacks the decryption key.

However, it is non-trivial task to apply FHE to NNs to construct Homomorphic NNs, and the process faces multiple challenges. First, many mainstream FHE schemes, such as BGV [263] and BFV [264] schemes, do not natively support float point arithmetic which are often found in neural network computations. Second, applying FHE to neural networks usually incur large performance and resource overheads in terms of computation and memory demands. This is attributed to the fact that plain data expands into large polynomials during homomorphic computation, and simple operations, like multiplications, translate into complex polynomial operations homomorphically [265]. Third, deeper networks involve deeper computations, which requires larger FHE parameters or even the expensive bootstrapping operations to control the noise growth in ciphertexts, which further increases the computationally intensity. Fortunately, neural network compression techniques can help to alleviate these challenges. This section reviews how NN compression helps on HNNs.

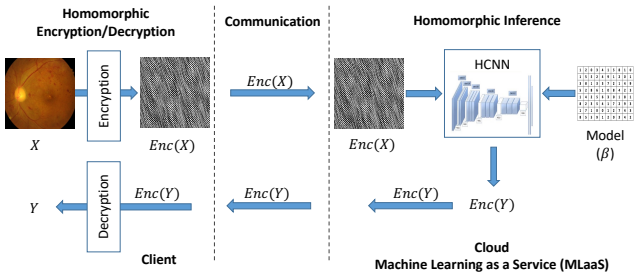

Fig. 2. Homomorphic neural network for secure cloud computing in Secure Machine Learning as a Service scenario.

B. NN Compression for HNNs

Various model compression techniques discussed in Section II, such as model quantization, pruning, and low-rank factorization, are commonly employed to enhance the inference speed of Homomorphic Neural Networks (HNNs). For instance, in [266], Lloyd-max quantization was utilized to quantize weights based on their local density during training. In [267], log-quantization was employed to transform real-valued data into power-of-two values, effectively converting matrix multiplication operations into faster shift operations to accelerate inference. Some approaches concentrate on binarizing the network to expedite HNNs inference, as observed in works like [268] and [269]. TAPAS [269], for instance, optimized FHE-based encrypted predictions by leveraging binary networks. Furthermore, specific strategies adopt network pruning to diminish inference latency. For example, CryptoNets [270] applied unstructured weight pruning to reduce the inference latency of Homomorphic neural networks, bypassing homomorphic operations related to pruned weights. Some techniques amalgamate quantization and pruning to expedite inference, as exemplified in [270], which proposes both pruning and quantization of models to minimize the required operations and enhance weight sparsity. Another

avenue explores low-rank factorization techniques. A recent approach presented in [271] integrates low-rank factorization with FHE ciphertext packing, effectively mitigating rotation overhead and significantly accelerating the inference process.

C. Hardware for HNNs

Researchers have proposed acceleration techniques for HNNs on various hardware platforms, *e.g.*, GPUs, Multi-core CPUs, FPGAs, and ASICs. Typically, hardware aids HNNs by speeding up HE operations, as demonstrated in various works [272], [272]–[274]. In particular, [273] introduced an FPGA-based computation accelerator within a Homomorphic Encryption Processing Unit (HEPU) co-processor, mitigating computational bottlenecks in lattice encryption primitives to improve the practicality of computing on encrypted data. Another approach involved optimizing hardware costs, including memory usage and energy consumption, as explored in works such as [265], [275], [276]. For instance, [265] proposed a cost-effective chiplet-based FHE implementation with a non-blocking inter-chiplet communication strategy to reduce data exchange overhead. Moreover, [277] achieved the first full realization of FHE in hardware, while [278] presented a custom hardware accelerator architecture combining algorithmic optimizations to accelerate server-side HE inference toward plaintext speeds.

D. Challenges in Compressing HNNs

Performing compression on HNNs presents significant challenges due to the computational and operational complexity of homomorphic encryption, in which multiple factors needed to be taken into consideration, such as the homomorphic encryption schemes, and the ciphertext packing strategies. Notably, neural network quantization works well with bit-wise TFHE schemes [279] or integer-based BFV schemes [264], but not with CKKS schemes [280] which encodes float-point numbers. On the other hand, neuron or filter based pruning may not work well with packed weights or activations in ciphertexts. Compatibility issues with homomorphic encryption libraries and the absence of standardization create obstacles in achieving effective compression. Researchers are actively working on overcoming these challenges to make compression in HNNs more practical for real-world applications.

V. Challenges, Future Trends and Conclusions

In this article, we have summarized recent works on compressing and accelerating DNNs. Next, we discuss some potential challenges and future trends.

Generalization While most state-of-the-art compression approaches are designed or verified on CNNs, their performance on other types of networks, especially larger and more complex models such as GPT-3 [281] and LLaMA [282] with billion parameters, and tiny models is not well established. For example, to reduce the cost of training complex models, post-quantization [62], [283] has been explored to quantize the weights and activations after the model is trained. [284] further quantizes tiny models ($\leq$ 1MB) in aggressive low-bit while retaining the accuracy.

Besides, most latest approaches focus on 2D data, which limits their real applications to handle other data, such as 3D and time-series data. 3D models can be highly complex, with millions of polygons and texture maps. Compressing such models can be a challenging task that requires specialized algorithms and hardware.

Furthermore, due to the existence of domain discrepancy between the model development and deployment stage, compressed models trained on historical data may suffer from performance degradation at the deployment stage. Although some existing works [285], [286] have already attempted to simultaneously address domain shift while compressing the model with knowledge distillation, cross-domain model compression is still not well explored.

Evaluation Although evaluation metrics (FLOPs, Model size, Latency) have been proposed for evaluating model compression, two drawbacks still exist. First, no standard hardware exists to evaluate various compression techniques. Consequently, many studied focus on theoretical analysis using FLOPs and model size. However, the cost of custom hardware is also affected by its memory and computation capability. Second, some works use outdated or non-standard hardware for evaluation, making it difficult to perform a fair comparison when conducting experiments on different hardware.

Hardware-software co-design Hardware constraints in various platforms, such as mobile and IoT chips, pose a significant challenge for the development of DNNs. Designing specific compression techniques for these platforms remains a critical bottleneck. Each platform would provide different operating conditions in energy per operation, memory-access latency, memory capacity, etc. Thus, selecting the compression mechanism should be *energy aware* to maximize the gains of applying such techniques. Besides, fully using the existing compression approaches is critical. For example, efficiently deploying the sparse architecture of an unstructured pruning network to custom hardware remains a challenge.

Integration Many compression approaches, like model pruning, quantization and knowledge distillation (KD), can be integrated together as they compress the DNNs from different perspectives. For example, there is a growing trend to combine KD with quantization and pruning.

Heterogeneous resource management Considering the workload and network conditions, it is necessary to distribute DNN processing tasks across heterogeneous computational platforms, including CPU, GPU, the cloud, and smartphones. In this way, the system can remain responsive to near real-time needs. However, how to connect and collaborate with different devices/platforms remains unexplored. Overall, efficient heterogeneous resource management is critical for the successful deployment of DNNs on a range of platforms. Continued research in this area can help optimize the performance of DNNs while minimizing resource usage.

Finally, our survey offers a comprehensive and unique perspective on neural network compression from three different angles: model compression, hardware accelerators, and homomorphic encryption. To the best of our knowledge, this is the first survey paper that comprehensively investigates this problem from software, hardware, and security perspec-

tives. By exploring existing techniques and potential future improvements, we aim to encourage the broader adoption of these approaches in real-world applications. Such adoption can create significant economic and social benefits by making AI more sustainable and cost-efficient in secure environments that save memory, energy, and computation. We believe that our survey can provide valuable information for researchers looking to address critical research problems and further advance the field for the benefit of society.

VI. Acknowledgements

This research is supported by the Agency for Science, Technology, and Research (A*STAR) under its Funds (AME Programmatic Funds A1892b0026, A19E3b0099, MTC Programmatic Fund M23L7b0021, NRF AME Young Individual Research Grant A2084c0167 and Career Development Fund C210812035). However, any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of the A*STAR.

References

[1] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, E. Li, X. Wang, M. Dehghani, S. Brahma *et al.*, "Scaling instruction-finetuned language models," *arXiv preprint arXiv:2210.11416*, 2022.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, 2017.

[3] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *IEEE SmartCloud*, 2016.

[4] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, 2020.

[5] R. Mishra and H. Gupta, "Transforming large-size to lightweight deep neural networks for iot applications," *ACM Computing Surveys*, 2023.

[6] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, 2018.

[7] R. Mishra, H. P. Gupta, and T. Dutta, "A survey on deep neural network compression: Challenges, overview, and solutions," *arXiv preprint arXiv:2010.03954*, 2020.

[8] J. O. Neill, "An overview of neural network compression," *arXiv preprint arXiv:2006.03669*, 2020.

[9] A. Alqahtani, X. Xie, and M. W. Jones, "Literature review of deep network compression," in *Informatics*. Multidisciplinary Digital Publishing Institute, 2021.

[10] A. Berthelier, T. Chateau, S. Duffner, C. Garcia, and C. Blanc, "Deep model compression and architecture optimization for embedded systems: A survey," *Journal of Signal Processing Systems*, 2021.

[11] S. Mittal and S. Umesh, "A survey on hardware accelerators and optimization techniques for rnns," *Journal of Systems Architecture*, 2021.

[12] K. Nan, S. Liu, J. Du, and H. Liu, "Deep model compression for mobile platforms: A survey," *Tsinghua Science and Technology*, 2019.

[13] C. Xu and J. McAuley, "A survey on model compression for natural language processing," *arXiv preprint arXiv:2202.07105*, 2022.

[14] M. Gupta and P. Agrawal, "Compression of deep learning models for text: A survey," *TKDD*, 2022.

[15] Y. Guo, "A survey on methods and theories of quantized neural networks," *arXiv preprint arXiv:1808.04752*, 2018.

[16] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[17] S. Xu, A. Huang, L. Chen, and B. Zhang, "Convolutional neural network pruning: A survey," in *CCC*, 2020.

[18] J. Liu, S. Tripathi, U. Kurup, and M. Shah, "Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey," *arXiv preprint arXiv:2005.04275*, 2020.

[19] R. Reed, "Pruning algorithms-a survey," *IEEE TNNLS*, 1993.

[20] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *JMLR*, 2019.

[21] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *CSUR*, 2021.

[22] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," *arXiv preprint arXiv:1905.01392*, 2019.

[23] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *IJCV*, 2021.

[24] L. Wang and K.-J. Yoon, "Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks," *T-PAMI*, 2021.

[25] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[26] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: learned quantization for highly accurate and compact deep neural networks," in *ECCV*, 2018.

[27] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," in *arXiv preprint arXiv:1702.03044*, 2017.

[28] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," in *arXiv preprint arXiv:1606.06160*, 2016.

[29] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," in *NIPS Workshop*, 2016.

[30] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," in *ICLR*, 2015.

[31] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *ICML*, 2015.

[32] W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing convolutional neural networks in the frequency domain," in *SIGKDD*, 2016.

[33] P. Stock, A. Joulin, R. Gribonval, B. Graham, and H. Jegou, "And the bit goes down: revisiting the quantization of neural networks," in *ICLR*, 2020.

[34] Y. Li, X. Dong, and W. Wang, "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks," in *ICLR*, 2020.

[35] J. Faraone, N. Fraser, M. Blott, and P. H. Leong, "Syq: Learning symmetric quantization for efficient deep neural networks," in *arXiv*, 2018.

[36] A. T. Elthakeb, P. Pilligundla, F. Mireshghallah, A. Yazdanbakhsh, and H. Esmaeilzadeh, "Releq: A reinforcement learning approach for deep quantization of neural networks," in *NeurIPS Workshop on ML for Systems*, 2018.

[37] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *CVPR*, 2019.

[38] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of convnets via differentiable neural architecture search," in *ICLR*, 2019.

[39] S. Uhlich, L. Mauch, F. Cardinaux, K. Yoshiyama, J. A. Garcia, S. Tiedemann, T. Kemp, and A. Nakamura, "Mixed precision dnns: All you need is a good parametrization," *arXiv preprint arXiv:1905.11452*, 2019.

[40] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *arXiv*, 2019.

[41] Z. Qu, Z. Zhou, Y. Cheng, and L. Thiele, "Adaptive loss-aware quantization for multi-bit networks," in *arXiv*, 2020.

[42] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry, "Post-training 4-bit quantization of convolution networks for rapid-deployment," *arXiv preprint arXiv:1810.05723*, 2018.

[43] L. Yang and Q. Jin, "Fracbits: Mixed precision quantization via fractional bit-widths," *arXiv preprint arXiv:2007.02017*, 2020.

[44] S. Zhao, T. Yue, and X. Hu, "Distribution-aware adaptive multi-bit quantization," in *CVPR*, 2021.

[45] Q. Lou, F. Guo, M. Kim, L. Liu, and L. Jiang, "Autoq: Automated kernel-wise neural network quantizations," in *ICLR*, 2020.

[46] A. Dubey, M. Chatterjee, and N. Ahuja, "Coreset-based neural network compression," in *arXiv:1807.09810*, 2018.

[47] S. W. et al., "Deepcabac: Context-adaptive binary arithmetic coding for deep neural network compression," in *ICML Workshop*, 2019.

[48] D. Oktay, J. Ballé, S. Singh, and A. Shrivastava, "Scalable model compression by entropy penalized reparameterization," in *ICLR*, 2020.

[49] W. Zhe, J. Lin, M. S. Aly, S. Young, V. Chandrasekhar, and B. Girod, "Rate-distortion optimized coding for efficient cnn compression," in *DCC*, 2021.

[50] S. Khoram and J. Li, "Adaptive quantization of neural networks," in *ICLR*, 2018.

[51] S. Young, Z. Wang, D. Taubman, and B. Girod, "Transform quantization for cnn compression," *T-PAMI*, 2021.

[52] Y. Wang, C. Xu, S. You, D. Tao, and C. Xu, "Cnnpack: packing convolutional neural networks in the frequency domain," in *NIPS*, 2016.

[53] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *NIPS*, 2014.

[54] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," *CVPR*, 2015.

[55] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *ICLR*, 2016.

[56] Y. Li, S. Gu, L. V. Gool, and R. Timofte, "Learning filter basis for convolutional neural network compression," *ICCV*, 2019.

[57] C. Li, T. Tang, G. Wang, J. Peng, B. Wang, X. Liang, and X. Chang, "Bossnas: Exploring hybrid cnn-transformers with block-wisely self-supervised neural architecture search," in *ICCV*, 2021.

[58] M. Chen, H. Peng, J. Fu, and H. Ling, "Autoformer: Searching transformers for visual recognition," in *ICCV*, 2021.

[59] Y. Bondarenko, M. Nagel, and T. Blankevoort, "Understanding and overcoming the challenges of efficient transformer quantization," in *EMNLP*, 2021.

[60] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale," *BIPS*, 2022.

[61] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao, "Post-training quantization for vision transformer," *NIPS*, 2021.

[62] Y. Ding, H. Qin, Q. Yan, Z. Chai, J. Liu, X. Wei, and X. Liu, "Towards accurate post-training quantization for vision transformer," in *ACM MM*, 2022.

[63] D. S. Taubman and M. W. Marcellin, "Jpeg2000 image compression fundamentals, standards and practice," *Journal of Electronic Imaging*, 2002.

[64] B. P. Tunstall, "Synthesis of noiseless compression codes," in *PhD diss., Georgia Institute of Technology*, 1967.

[65] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," in *ICLR*, 2018.

[66] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," in *ICLR*, 2020.

[67] P. de Jorge, A. Sanyal, H. Behl, P. Torr, G. Rogez, and P. K. Dokania, "Progressive skeletonization: Trimming more fat from a network at initialization," in *ICLR*, 2021.

[68] H. Wang, C. Qin, Y. Bai, Y. Zhang, and Y. Fu, "Recent advances on neural network pruning at initialization," in *IJCAI*, 2022.

[69] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *NIPS*, 1990.

[70] S. P. Singh and D. Alistarh, "Woodfisher: Efficient second-order approximation for neural network compression," in *NIPS*, 2020.

[71] E. Frantar, E. Kurtic, and D. Alistarh, "M-FAC: Efficient matrix-free approximations of second-order information," in *NIPS*, 2021.

[72] M. Shen, H. Yin, P. Molchanov, L. Mao, J. Liu, and J. M. Alvarez, "Structural pruning via latency-saliency knapsack," *NIPS*, 2022.

[73] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *CVPR*, 2019.

[74] K. Xu, Z. Wang, X. Geng, M. Wu, X. Li, and W. Lin, "Efficient joint optimization of layer-adaptive weight pruning in deep neural networks," in *ICCV*, 2023.

[75] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi, "Dynamic model pruning with feedback," in *ICLR*, 2020.

[76] J. Liu, Z. XU, R. SHI, R. C. C. Cheung, and H. K. So, "Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers," in *ICLR*, 2020.

[77] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *ICLR*, 2017.

[78] Y. Tang, Y. Wang, Y. Xu, D. Tao, C. Xu, C. Xu, and C. Xu, "Scop: Scientific control for reliable neural network pruning," *NIPS*, 2020.

[79] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi, "Soft threshold weight reparameterization for learnable sparsity," in *ICML*, 2020.

[80] P. Savarese, H. Silva, and M. Maire, "Winning the lottery with continuous sparsification," in *NIPS*, 2020.

[81] H. Wang, C. Qin, Y. Zhang, and Y. Fu, "Neural pruning via growing regularization," in *ICLR*, 2021.

[82] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, "Rigging the lottery: Making all tickets winners," in *ICML*, 2020.

[83] S. Park, J. Lee, S. Mo, and J. Shin, "Lookahead: a far-sighted alternative of magnitude-based pruning," in *ICLR*, 2020.

[84] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," in *ICLR Workshop*, vol. abs/1710.01878, 2018.

[85] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin, "Layer-adaptive sparsity for the magnitude-based pruning," in *ICLR*, 2021.

[86] M. Gupta, E. Camci, V. R. Keneta, A. Vaidyanathan, R. Kanodia, C.-S. Foo, W. Min, and L. Jie, "Is complexity required for neural network pruning? a case study on global magnitude pruning," *arXiv preprint arXiv:2209.14624*, 2022.

[87] H. Mostafa and X. Wang, "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization," in *ICML*, 2019.

[88] T. Dettmers and L. Zettlemoyer, "Sparse networks from scratch: Faster training without losing performance," *CoRR*, vol. abs/1907.04840, 2019.

[89] M. Gupta, S. Aravindan, A. Kalisz, V. Chandrasekhar, and L. Jie, "Learning to prune deep neural networks via reinforcement learning," *ICML AutoML Workshop*, 2020.

[90] He, Yihui, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *ECCV*, 2018.

[91] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *NIPS*, 2017.

[92] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *ICLR*, 2019.

[93] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," *NIPS*, 2019.

[94] E. Frantar and D. Alistarh, "Optimal brain compression: A framework for accurate post-training quantization and pruning," *NIPS*, 2022.

[95] W. Kwon, S. Kim, M. W. Mahoney, J. Hassoun, K. Keutzer, and A. Gholami, "A fast post-training pruning framework for transformers," *NIPS*, 2022.

[96] M. Wortsman, A. Farhadi, and M. Rastegari, "Discovering neural wirings," in *NIPS*, 2019.

[97] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *CVPR*, 2019.

[98] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *CVPR*, 2020.

[99] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu, "Pruning from scratch," in *AAAI*, 2020.

[100] Y. Li, S. Gu, K. Zhang, L. Van Gool, and R. Timofte, "Dhp: Differentiable meta pruning via hypernetworks," in *ECCV*, 2020.

[101] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann, "Towards optimal structured cnn pruning via generative adversarial learning," in *CVPR*, 2019.

[102] W. Wang, M. Chen, S. Zhao, L. Chen, J. Hu, H. Liu, D. Cai, X. He, and W. Liu, "Accelerate cnns from three dimensions: A comprehensive pruning framework," in *ICML*, 2021.

[103] F. Lagunas, E. Charlaix, V. Sanh, and A. M. Rush, "Block pruning for faster transformers," in *EMNLP*, 2021.

[104] A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, and H. Li, "Learning n: m fine-grained structured sparse neural networks from scratch," *arXiv preprint arXiv:2102.04010*, 2021.

[105] C. Fang, A. Zhou, and Z. Wang, "An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers," *IEEE T-VLSI*, 2022.

[106] H. Yang, H. Yin, M. Shen, P. Molchanov, H. Li, and J. Kautz, "Global vision transformer pruning with hessian-aware saliency," in *CVPR*, 2023.

[107] C. Yu, T. Chen, Z. Gan, and J. Fan, "Boost vision transformer with gpu-friendly sparsity and quantization," in *CVPR*, 2023.

[108] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations," *arXiv preprint arXiv:2308.06767*, 2023.

[109] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *ICLR*, 2018.

[110] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *ICCV*, 2019.

[111] C. Li, G. Wang, B. Wang, X. Liang, Z. Li, and X. Chang, "Dynamic slimmable network," in *CVPR*, 2021.

[112] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *ICLR*, 2020.

[113] C. Wang, X. Lan, and Y. Zhang, "Model distillation with knowledge transfer from face classification to alignment and verification," *arXiv preprint arXiv:1709.02929*, 2017.

[114] H. J. Lee, W. J. Baddar, H. G. Kim, S. T. Kim, and Y. M. Ro, "Teacher and student joint learning for compact facial landmark detection network," in *MMM*, 2018.

[115] Y. Zhu and Y. Wang, "Student customized knowledge distillation: Bridging the gap between student and teacher," in *ICCV*, 2021.

[116] D. Y. Park, M.-H. Cha, D. Kim, B. Han *et al.*, "Learning student-friendly teacher networks for knowledge distillation," in *NIPS*, 2021.

[117] Y. Kim and A. M. Rush, "Sequence-level knowledge distillation," *arXiv preprint arXiv:1606.07947*, 2016.

[118] M. A. Gordon and K. Duh, "Explaining sequence-level knowledge distillation as data-augmentation for neural machine translation," *arXiv preprint arXiv:1912.03334*, 2019.

[119] Z.-R. Wang and J. Du, "Joint architecture and knowledge distillation in cnn for chinese text recognition," *Pattern Recognition*, 2021.

[120] Y. Chebotar and A. Waters, "Distilling knowledge from ensembles of neural networks for speech recognition." in *Interspeech*, 2016.

[121] K. Kwon, H. Na, H. Lee, and N. S. Kim, "Adaptive knowledge distillation based on entropy," in *ICASSP*, 2020.

[122] Z. Li, Y. Ming, L. Yang, and J.-H. Xue, "Mutual-learning sequence-level knowledge distillation for automatic speech recognition," *Neurocomputing*, 2021.

[123] Q. Xu, Z. Chen, K. Wu, C. Wang, M. Wu, and X. Li, "Kdnet-rul: A knowledge distillation framework to compress deep neural networks for machine remaining useful life prediction," *IEEE Transactions on Industrial Electronics*, 2021.

[124] Q. Xu, Z. Chen, M. Ragab, C. Wang, M. Wu, and X. Li, "Contrastive adversarial knowledge distillation for deep model compression in time-series regression tasks," *Neurocomputing*, 2022.

[125] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[126] C. Zhang and Y. Peng, "Better and faster: knowledge transfer from multiple self-supervised learning tasks via graph distillation for video classification," in *IJCAI*, 2018.

[127] B. Zhao, Q. Cui, R. Song, Y. Qiu, and J. Liang, "Decoupled knowledge distillation," in *CVPR*, 2022.

[128] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," in *NIPS*, 2017.

[129] C. Yuan and R. Pan, "Obtain dark knowledge via extended knowledge distillation," in *AIAM*, 2019.

[130] S. Hegde, R. Prasad, R. Hebbalaguppe, and V. Kumar, "Variational student: Learning compact and sparser networks in knowledge distillation framework," in *ICASSP*, 2020.

[131] M. R. U. Saputra, P. P. De Gusmao, Y. Almalioglu, A. Markham, and N. Trigoni, "Distilling knowledge from a deep pose regressor network," in *ICCV*, 2019.

[132] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," in *ICLR*, 2017.

[133] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *CVPR*, 2017.

[134] J. Kim, S. Park, and N. Kwak, "Paraphrasing complex network: Network compression via factor transfer," in *NIPS*, 2018.

[135] S. Ahn, S. X. Hu, A. Damianou, N. D. Lawrence, and Z. Dai, "Variational information distillation for knowledge transfer," in *CVPR*, 2019.

[136] L. Liu, Q. Huang, S. Lin, H. Xie, B. Wang, X. Chang, and X. Liang, "Exploring inter-channel correlation for diversity-preserved knowledge distillation," in *ICCV*, 2021.

[137] S. Lin, H. Xie, B. Wang, K. Yu, X. Chang, X. Liang, and G. Wang, "Knowledge distillation via the target-aware transformer," in *CVPR*, 2022.

[138] Y. Shang, B. Duan, Z. Zong, L. Nie, and Y. Yan, "Lipschitz continuity guided knowledge distillation," in *ICCV*, 2021.

[139] Z. Huang, X. Shen, J. Xing, T. Liu, X. Tian, H. Li, B. Deng, J. Huang, and X.-S. Hua, "Revisiting knowledge distillation: An inheritance and exploration framework," in *CVPR*, 2021.

[140] M. Ji, B. Heo, and S. Park, "Show, attend and distill: Knowledge distillation via attention-based feature matching," in *AAAI*, 2021.

[141] N. Passalis and A. Tefas, "Learning deep representations with probabilistic knowledge transfer," in *ECCV*, 2018.

[142] W. Park, D. Kim, Y. Lu, and M. Cho, "Relational knowledge distillation," in *CVPR*, 2019.

[143] Y. Liu, J. Cao, B. Li, C. Yuan, W. Hu, Y. Li, and Y. Duan, "Knowledge distillation via instance relationship graph," in *CVPR*, 2019.

[144] T. Huang, S. You, F. Wang, C. Qian, and C. Xu, "Knowledge distillation from a stronger teacher," in *NIPS*, 2022.

[145] S. Yun, J. Park, K. Lee, and J. Shin, "Regularizing class-wise predictions via self-knowledge distillation," in *CVPR*, 2020.

[146] Y. Tian, D. Krishnan, and P. Isola, "Contrastive representation distillation," in *ICLR*, 2020.

[147] J. Zhu, S. Tang, D. Chen, S. Yu, Y. Liu, M. Rong, A. Yang, and X. Wang, "Complementary relation contrastive distillation," in *CVPR*, 2021.

[148] B. Peng, X. Jin, J. Liu, D. Li, Y. Wu, Y. Liu, S. Zhou, and Z. Zhang, "Correlation congruence for knowledge distillation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5007–5016.

[149] X. Geng, J. Lin, B. Zhao, A. Kong, M. M. S. Aly, and V. Chandrasekhar, "Hardware-aware softmax approximation for deep neural networks," in *ACCV*, 2018.

[150] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *CVPR*, 2005.

[151] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Soft-nms–improving object detection with one line of code," in *ICCV*, 2017.

[152] L. Tychsen-Smith and L. Petersson, "Improving object localization with fitness nms and bounded iou loss," in *CVPR*, 2018.

[153] B. Jiang, R. Luo, J. Mao, T. Xiao, and Y. Jiang, "Acquisition of localization confidence for accurate object detection," in *ECCV*, 2018.

[154] N. Gählert, N. Hanselmann, U. Franke, and J. Denzler, "Visibility guided nms: Efficient boosting of amodal object detection in crowded traffic scenes," *arXiv preprint arXiv:2006.08547*, 2020.

[155] Y. He, C. Zhu, J. Wang, M. Savvides, and X. Zhang, "Bounding box regression with uncertainty for accurate object detection," in *CVPR*, 2019.

[156] S. Liu, D. Huang, and Y. Wang, "Adaptive nms: Refining pedestrian detection in a crowd," in *CVPR*, 2019.

[157] N. O. Salscheider, "Featurenms: Non-maximum suppression by learning feature embeddings," in *ICPR*, 2021.

[158] J. Hosang, R. Benenson, and B. Schiele, "A convnet for non-maximum suppression," in *German Conference on Pattern Recognition*, 2016.

[159] J. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," in *CVPR*, 2017.

[160] H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei, "Relation networks for object detection," in *CVPR*, 2018.

[161] R. Rothe, M. Guillaumin, and L. V. Gool, "Non-maximum suppression for object detection by passing messages between windows," in *ACCV*, 2014.

[162] D. Oro *et al.*, "Work-Efficient Parallel Non-Maximum Suppression Kernels," *The Computer Journal*, 08 2020.

[163] C. Chen, T. Zhang, Z. Yu, A. Raghuraman, S. Udayan, J. Lin, and M. M. S. Aly, "Scalable hardware acceleration of non-maximum suppression," in *DATE*, 2022.

[164] L. Cai *et al.*, "Maxpoolnms: getting rid of nms bottlenecks in two-stage object detectors," in *CVPR*, 2019.

[165] T. Zhang *et al.*, "Psrr-maxpoolnms: Pyramid shifted maxpoolnms with relationship recovery," in *CVPR*, 2021.

[166] A. Joulin, M. Cissé, D. Grangier, H. Jégou *et al.*, "Efficient softmax approximation for gpus," in *ICML*, 2017.

[167] K. Shim, M. Lee, I. Choi, Y. Boo, and W. Sung, "Svd-softmax: Fast softmax approximation on large vocabulary neural networks," in *NIPS*, 2017.

[168] G. Blanc and S. Rendle, "Adaptive sampled softmax with kernel based sampling," in *ICML*, 2018.

[169] J. Lu, J. Yao, J. Zhang, X. Zhu, H. Xu, W. Gao, C. Xu, T. Xiang, and L. Zhang, "Soft: Softmax-free transformer with linear complexity," in *NIPS*, 2021.

[170] K. Zhao, L. Song, Y. Zhang, P. Pan, Y. Xu, and R. Jin, "Ann softmax: acceleration of extreme classification training," *Proceedings of the VLDB Endowment*, 2021.

[171] P. H. Chen, S. Si, S. Kumar, Y. Li, and C.-J. Hsieh, "Learning to screen for fast softmax inference on large vocabulary neural networks," *arXiv preprint arXiv:1810.12406*, 2018.

[172] S. Liao, T. Chen, T. Lin, D. Zhou, and C. Wang, "Doubly sparse: Sparse mixture of sparse experts for efficient softmax inference," *arXiv preprint arXiv:1901.10668*, 2019.

[173] A. S. Rawat, J. Chen, F. X. X. Yu, A. T. Suresh, and S. Kumar, "Sampled softmax with random fourier features," in *NIPS*, 2019.

[174] J. Andreas, M. Rabinovich, M. I. Jordan, and D. Klein, "On the accuracy of self-normalized log-linear models," *NIPS*, 2015.

[175] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *NIPS*, 2012.

[176] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015.

[177] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, 2019.

[178] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[179] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *CVPR*, 2019.

[180] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *CVPR*, 2019.

[181] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *ICLR*, 2019.

[182] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *ICML*, 2018.

[183] A. Ancilotto, F. Paissan, and E. Farella, "Xinet: Efficient neural networks for tinyml," in *ICCV*, 2023.

[184] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "Mcunetv2: Memory-efficient patch-based inference for tiny deep learning," *arXiv preprint arXiv:2110.15352*, 2021.

[185] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce memory, not parameters for efficient on-device learning," *NIPS*, 2020.

[186] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, 2014.

[187] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *MICRO*, 2014.

[188] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," *ACM SIGARCH Computer Architecture News*, 2015.

[189] Nvidia. (2018) NVIDIA Deep Learning Accelerator. [Online]. Available: http://nvdla.org/primer.html

[190] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *ACM SIGPLAN Notices*, 2018.

[191] V. Strassen, "Gaussian elimination is not optimal," *Numerische mathematik*, 1969.

[192] S. Winograd, *Arithmetic complexity of computations*. Siam, 1980.

[193] E. O. Brigham and R. Morrow, "The fast fourier transform," *IEEE spectrum*, 1967.

[194] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *ICANN*, 2014.

[195] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *CVPR*, 2016.

[196] S. Ben-Yacoub, "Fast object detection using mlp and fft," IDIAP, Tech. Rep., 1997.

[197] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," *arXiv preprint arXiv:1312.5851*, 2013.

[198] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," *arXiv preprint arXiv:1412.7580*, 2014.

[199] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on fpgas," *IEEE TCAD*, 2019.

[200] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang, "Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer," in *SOCC*, 2020.

[201] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee *et al.*, "Aˆ 3: Accelerating attention mechanisms in neural networks with approximation," in *HPCA*, 2020.

[202] W. Hu, D. Xu, Z. Fan, F. Liu, and Y. He, "Vis-top: Visual transformer overlay processor," *arXiv preprint arXiv:2110.10957*, 2021.

[203] M. Sun, H. Ma, G. Kang, Y. Jiang, T. Chen, X. Ma, Z. Wang, and Y. Wang, "Vaqf: Fully automatic software-hardware co-design framework for low-bit vision transformer," *arXiv preprint arXiv:2201.06618*, 2022.

[204] J. Yu, J. Park, S. Park, M. Kim, S. Lee, D. H. Lee, and J. Choi, "Nn-lut: neural approximation of non-linear operations for efficient transformer inference," in *DAC*, 2022.

[205] A. Marchisio, D. Dura, M. Capra, M. Martina, G. Masera, and M. Shafique, "Swifttron: An efficient hardware accelerator for quantized transformers," *arXiv preprint arXiv:2304.03986*, 2023.

[206] S. Nag, G. Datta, S. Kundu, N. Chandrachoodan, and P. A. Beerel, "Vita: A vision transformer inference accelerator for edge applications," *arXiv preprint arXiv:2302.09108*, 2023.

[207] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, L. Liu, and S. Wei, "A 1.06-to-5.09 tops/w reconfigurable hybrid-neural-network processor for deep learning applications," in *Symposium on VLSI Circuits*, 2017.

[208] H. Zhang, D. Chen, and S.-B. Ko, "New flexible multiple-precision multiply-accumulate unit for deep neural network training and inference," *IEEE Transactions on Computers*, 2019.

[209] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 lnpu: A 25.3 tflops/w sparse deep-neural-network learning processor with fine-grained mixed precision of fp8-fp16," in *ISSCC*. IEEE, 2019.

[210] X. Zhou, L. Zhang, C. Guo, X. Yin, and C. Zhuo, "A convolutional neural network accelerator architecture with fine-granular mixed precision configurability," in *ISCAS*, 2020.

[211] B. Fleischer, S. Shukla, M. Ziegler, J. Silberman, J. Oh, V. Srinivasan, J. Choi, S. Mueller, A. Agrawal, T. Babinsky *et al.*, "A scalable multi-teraops deep learning processor core for ai trainina and inference," in *IEEE Symposium on VLSI Circuits*, 2018.

[212] J. Wang, Q. Lou, X. Zhang, C. Zhu, Y. Lin, and D. Chen, "Design flow of accelerating hybrid extremely low bit-width neural network in embedded fpga," in *FPL*, 2018.

[213] A. Agrawal, S. K. Lee, J. Silberman, M. Ziegler, M. Kang, S. Venkataramani, N. Cao, B. Fleischer, M. Guillorn, M. Cohen *et al.*, "9.1 a 7nm 4-core ai chip with 25.6 tflops hybrid fp8 training, 102.4 tops int4 inference and workload-aware throttling," in *ISSCC*, 2021.

[214] S. Recanatesi *et al.*, "Dimensionality compression and expansion in deep neural networks," *arXiv preprint arXiv:1906.00443*, 2019.

[215] D. Oktay *et al.*, "Scalable model compression by entropy penalized reparameterization," *arXiv preprint arXiv:1906.06624*, 2019.

[216] S. Wiedemann *et al.*, "Deepcabac: Context-adaptive binary arithmetic coding for deep neural network compression," *arXiv preprint arXiv:1905.08318*, 2019.

[217] A. Dubey *et al.*, "Coreset-based neural network compression," in *ECCV*, 2018.

[218] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[219] C. Chen, Z. Wang, X. Chen, J. Lin, and M. M. S. Aly, "Efficient tunstall decoder for deep neural network compression," in *DAC*, 2021.

[220] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[221] M. Rabbani, "Jpeg2000: Image compression fundamentals, standards and practice," *Journal of Electronic Imaging*, 2002.

[222] I. H. Witten *et al.*, "Arithmetic coding for data compression," *Communications of the ACM*, 1987.

[223] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, 2016.

[224] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *ISCA*, 2016.

[225] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *ISCA*, 2017.

[226] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[227] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *Proceedings of the IEEE*, 2021.

[228] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *MICRO*, 2016.

[229] F. G. Gustavson, "Some basic techniques for solving sparse systems of linear equations," in *Sparse matrices and their applications*, 1972.

[230] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "Splatt: Efficient and parallel sparse tensor-matrix multiplication," in *IPDPS*, 2015.

[231] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, "Extensor: An accelerator for sparse tensor algebra," in *MICRO*, 2019.

[232] Z. Yuan, J. Yue, H. Yang, Z. Wang, J. Li, Y. Yang, Q. Guo, X. Li, M.-F. Chang, H. Yang *et al.*, "Sticker: A 0.41-62.1 tops/w 8bit neural network processor with multi-sparsity compatible convolution arrays and online tuning acceleration for fully connected layers," in *IEEE symposium on VLSI circuits*, 2018.

[233] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang, "Snap: A 1.67—21.55 tops/w sparse neural acceleration processor for unstructured sparse deep neural network inference in 16nm cmos," in *VLSIC*. IEEE, 2019.

[234] H.-J. Kang, "Accelerator-aware pruning for convolutional neural networks," *IEEE TCSVT*, 2019.

[235] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," in *FCCM*, 2019.

[236] J. J. Zhang, P. Raj, S. Zarar, A. Ambardekar, and S. Garg, "Compact: On-chip compression of activations for low power systolic array based cnn acceleration," *TECS*, 2019.

[237] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *MICRO*, 2018.

[238] A. K. Mishra, E. Nurvitadhi, G. Venkatesh, J. Pearce, and D. Marr, "Fine-grained accelerators for sparse machine learning workloads," in *ASP-DAC*, 2017.

[239] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *ACM/SIGDA FPGA*, 2017.

[240] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *ISSCC*, 2017.

[241] D. Oro *et al.*, "Work-efficient parallel non-maximum suppression for embedded gpu architectures," in *ICASSP*, 2016.

[242] M. Shi *et al.*, "A fast and power-efficient hardware architecture for non-maximum suppression," *TCAS-II*, 2019.

[243] B. Yuan, "Efficient hardware architecture of softmax layer in deep neural network," in *SOCC*, 2016.

[244] Z. Li, H. Li, X. Jiang, B. Chen, Y. Zhang, and G. Du, "Efficient fpga implementation of softmax function for dnn applications," in *ASID*, 2018.

[245] M. Wang, S. Lu, D. Zhu, J. Lin, and Z. Wang, "A high-speed and low-complexity architecture for softmax function in deep learning," in *APCCAS*. IEEE, 2018.

[246] Q. Sun, Z. Di, Z. Lv, F. Song, Q. Xiang, Q. Feng, Y. Fan, X. Yu, and W. Wang, "A high speed softmax vlsi architecture based on basic-split," in *ICSICT*, 2018.

[247] R. Hu, B. Tian, S. Yin, and S. Wei, "Efficient hardware architecture of softmax layer in deep neural network," in *ICDSP*, 2018.

[248] G. Du, C. Tian, Z. Li, D. Zhang, Y. Yin, and Y. Ouyang, "Efficient softmax hardware architecture for deep neural networks," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019.

[249] I. Kouretas and V. Paliouras, "Hardware implementation of a softmax-like function for deep learning," *Technologies*, 2020.

[250] K.-Y. Wang, Y.-D. Huang, Y.-L. Ho, and W.-C. Fang, "A customized convolutional neural network design using improved softmax layer for real-time human emotion recognition," in *AICAS*, 2019.

[251] G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, A. Nannarelli, M. Re, and S. Spanò, "A pseudo-softmax function for hardware-based high speed image classification," *Scientific reports*, 2021.

[252] F. Spagnolo, S. Perri, and P. Corsonello, "Aggressive approximation of the softmax function for power-efficient hardware implementations," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[253] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, "Softermax: Hardware/software co-design of an efficient softmax for transformers," in *DAC*, 2021.

[254] B. R. Gaines, "Stochastic computing systems," *Advances in Information Systems Science*, 1969.

[255] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications," *IEEE TNNLS*, 2020.

[256] S. A. Salehi, "Low-cost stochastic number generators for stochastic computing," *VLSI*, 2020.

[257] C. F. Frasser, P. Linares-Serrano, I. D. de Los Rios, A. Moran, E. S. Skibinsky-Gitlin, J. Font-Rossello, V. Canals, M. Roca, T. Serrano-Gotarredona, and J. L. Rossello, "Fully parallel stochastic computing hardware implementation of convolutional neural networks for edge computing applications," *IEEE TNNLS*, 2022.

[258] A. Morán, L. Parrilla, M. Roca, J. Font-Rossello, E. Isern, and V. Canals, "Digital implementation of radial basis function neural networks based on stochastic computing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2022.

[259] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Eurocrypt*, 1999.

[260] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, 1978.

[261] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009.

[262] M. Isakov, V. Gadepally, K. M. Gettings, and M. A. Kinsy, "Survey of attacks and defenses on edge-deployed neural networks," in *HPEC*, 2019.

[263] M. Yagisawa, "Fully homomorphic encryption without bootstrapping," *Cryptology ePrint Archive*, 2015.

[264] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[265] A. Aikata, A. C. Mert, S. Kwon, M. Deryabin, and S. S. Roy, "Reed: Chiplet-based scalable hardware accelerator for fully homomorphic encryption," *arXiv preprint arXiv:2308.02885*, 2023.

[266] S.-X. Zhang, Y. Gong, and D. Yu, "Encrypted speech recognition using deep polynomial networks," in *ICASSP*, 2019.

[267] Q. Lou and L. Jiang, "She: A fast and accurate deep neural network for encrypted data," *NIPS*, 2019.

[268] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Crypto*, 2018.

[269] A. Sanyal, M. Kusner, A. Gascon, and V. Kanade, "TAPAS: Tricks to accelerate (encrypted) prediction as a service," in *ICML*, 2018.

[270] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *arXiv preprint arXiv:1811.09953*, 2018.

[271] Y. Lu, J. Lin, C. Jin, Z. Wang, M. Wu, K. M. M. Aung, and X. Li, "Ffconv: Fast factorized convolutional neural network inference on encrypted data," *arXiv preprint arXiv:2102.03494*, 2021.

[272] M. Khalil-Hani, V. P. Nambiar, and M. Marsono, "Hardware acceleration of openssl cryptographic functions for high-performance internet security," in *International Conference on Intelligent Systems, Modelling and Simulation*, 2010.

[273] D. B. Cousins, K. Rohloff, and D. Sumorok, "Designing an fpga-accelerated homomorphic encryption co-processor," *IEEE Transactions on Emerging Topics in Computing*, 2016.

[274] D. Lee, W. Lee, H. Oh, and K. Yi, "Optimizing homomorphic evaluation circuits by program synthesis and term rewriting," in *PLDI*, 2020.

[275] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "Sharp: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *ISCA*, 2023.

[276] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "Bts: An accelerator for bootstrappable fully homomorphic encryption," in *ISCA*, 2022.

[277] Y. Doröz, E. Öztürk, and B. Sunar, "Accelerating fully homomorphic encryption in hardware," *IEEE Transactions on Computers*, 2014.

[278] B. Reagen, W.-S. Choi, Y. Ko, V. T. Lee, H.-H. S. Lee, G.-Y. Wei, and D. Brooks, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," in *HPCA*, 2021.

[279] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, 2020.

[280] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT*, 2017.

[281] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *NIPS*, 2020.

[282] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[283] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large-scale transformers," *NIPS*, 2022.

[284] K. Xu, Y. Li, H. Zhang, R. Lai, and L. Gu, "Etinynet: Extremely tiny network for tinyml," in *AAAI*, 2022.

[285] J. Yang, H. Zou, S. Cao, Z. Chen, and L. Xie, "Mobileda: Toward edge-domain adaptation," *IEEE Internet of Things Journal*, 2020.
[286] M. Ryu, G. Lee, and K. Lee, "Knowledge distillation for bert unsupervised domain adaptation," *Knowledge and Information Systems*, 2022.