Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Composing Lightweight Methods on Deep Neural Network: A Survey

Kilho Son
Microsoft Corporation
Yunho Jeon
yhjeon@hanbat.ac.kr

Hanbat National University

Research Article

Keywords: Model Compression, Lightweight, Composition of Compression methods, Deep Learning, Neural Network

Posted Date: November 11th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11232383/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Composing Lightweight Methods on Deep Neural Network: A Survey

Kilho Son[1] and Yunho Jeon[2*]

[1]Microsoft Corporation, 1 Microsoft Way, Redmond, WA, 98052, United States.
[2]Hanbat National University, Sejong, 30139, Republic of Korea.

*Corresponding author(s). E-mail(s): yhjeon@hanbat.ac.kr;

Abstract

Deep learning has achieved remarkable progress across many domains, largely powered by large-scale neural networks. Their substantial computational demands, however, hinder deployment on resource-constrained edge devices. To address this, lightweight techniques, such as efficient architecture design, pruning, quantization, decomposition, and knowledge distillation, have been explored to reduce the required resources while preserving accuracy. While prior surveys typically investigate these methods one by one, limited attention has been given to how their composition, ordering, and optimization dependencies affect efficiency and accuracy in real-world settings. In this survey, we introduce a taxonomy of lightweight method compositions, organized by which lightweight methods are selected and how those methods are composed and interact, including sequential application, joint optimization, and hybrid approaches. We qualitatively and quantitatively analyze these categories, with particular attention to within-study comparisons, to assess the effectiveness of different compositions in improving efficiency and accuracy, providing guidance for future research in this area and practical deployment of deep neural networks.

Keywords: Model Compression, Lightweight, Composition of Compression methods, Deep Learning, Neural Network

1 Introduction

Deep learning has achieved significant performance gains across many fields, largely through the use of large neural networks (NNs). However, these models require

substantial memory and computational resources. Early-stage convolutional neural networks (CNNs) like VGG Simonyan and Zisserman (2015) and ResNet He et al. (2016) consist of 144 million (VGG-19) and 26 million (ResNet-50) parameters, respectively. Transformer-based models, such as GPT-3, contain 175 billion parameters Brown et al. (2020). These models often require gigabytes of memory and trillions of multiply-accumulate operations. As a result, operating them typically requires high-performance hardware on device or cloud servers.

As mobile phones, IoT devices, and other edge platforms become more widely used, the volume of generated and processed data continues to grow rapidly. Because limited computational resources on edge devices often make it difficult to run large NN models locally, data is commonly transmitted to a central server for processing and then returned to the edge device. This process, however, burdens network infrastructure through heavy data transfer, making it inadequate for applications that require real-time responses. Beyond technical issues, moving data to the server raises privacy issues. Transferring healthcare, financial, personal communication, or social-network data to other devices is often prohibited because such data may contain personal information, either directly or indirectly.

These challenges have increased the demand for technologies that perform computation directly on edge devices instead of relying on centralized servers. This paradigm, known as on-device AI, processes data locally on the device rather than transmitting it to the cloud. A key challenge in on-device AI is to optimize model size and resource requirements so that models can operate within strict hardware constraints. To address this, model compression and lightweight architectures have been extensively studied, with the goal of preserving or even enhancing the accuracy of deep learning models while reducing their complexity and computational requirements.

Lightweight methods can be broadly classified into five categories: efficient architecture design, decomposition, pruning, quantization, and knowledge distillation (See Figure 1). Many previous works have focused on improving efficiency using a single lightweight method. Several survey papers have explored individual methods in depth, providing a comprehensive overview of the current state of research in each area: lightweight architectures Li et al. (2021); Liu et al. (2017); Mi et al. (2022); Nan et al. (2019), pruning Liu et al. (2020); Augasta and Kathirvalavakumar (2013); Blalock et al. (2020); Cheng et al. (2024); He and Xiao (2023); Vadera and Ameen (2022); Xu et al. (2020), quantization Gholami et al. (2022); Guo (2018); Kulkarni et al. (2022); Mishra et al. (2020); Nagel et al. (2021); Rokh et al. (2023); Wei et al. (2024); Weng (2021), and knowledge distillation Gou et al. (2021); Kaleem et al. (2024); Wang and Yoon (2021). Many surveys also provide broader introductions covering several of these methods Cheng et al. (2017, 2018,?); Choudhary et al. (2020); Deng et al. (2020); Ghimire et al. (2022); Li et al. (2023); Liang et al. (2021); Mishra et al. (2020); Neill (2020); Sze et al. (2017); Chen et al. (2024).

In practical applications, developing an efficient AI model that meets resource constraints typically requires integrating multiple compression techniques. Composing lightweight methods often achieves better accuracy and lower computational cost than single method.

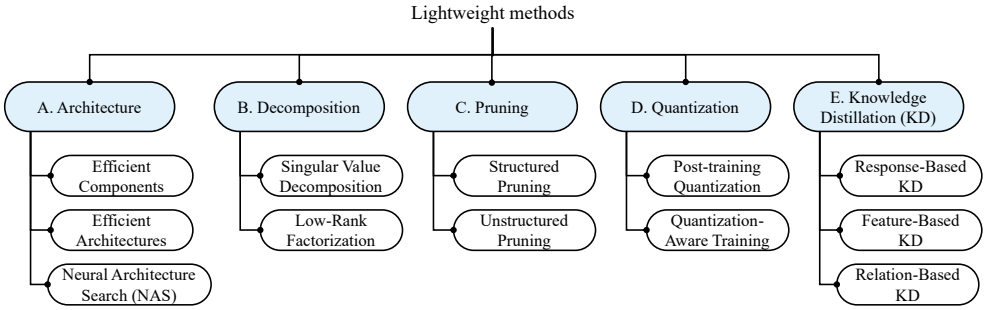


Fig. 1: A brief taxonomy of the individual lightweight methods.

Several existing surveys discuss the combination of multiple methods, but none use composition as the main organizing principle. Broad compression surveys, such as Deng et al. (2020) and the lightweight-CNN review of Chen et al. (2024), organize the literature by individual techniques and, in some cases, by hardware platforms. In these surveys, combined methods appear only as parts of larger compression or deployment pipelines, rather than as a dedicated taxonomy axis. Among them, the pruning survey of Cheng et al. (2024) is closest to our scope by covering combinations of pruning with other compression methods, although its analysis remains pruning-centric. The quantization survey of Gholami et al. (2022) and the knowledge distillation taxonomy of Gou et al. (2021) mention other compression methods mainly as supporting context or as application examples within a single-method framework. The language-model survey of Park et al. (2024) applies this technique-oriented view to Transformers and treats the combination of methods mainly as future work. Overall, existing surveys cover each individual method thoroughly. However, they rarely treat method order or optimization dependencies as primary dimensions and offer limited quantitative comparison of the compositions themselves.

The key feature of this survey is not just that it collects papers using multiple lightweight methods. It treats the composition itself as the object of study. Specifically, we classify compositions according to optimization dependencies, distinguish sequential, joint, and hybrid pipelines, examine how ordering and mutual influence produce complementary or conflicting behavior, and compare composition-level ablations reported in prior work. This perspective helps identify when a second method adds orthogonal compression, recovers damage from an earlier stage, or requires joint optimization rather than a staged pipeline. These questions are not systematically addressed in technique-centered surveys.

In this survey, we select studies from major conferences and journals in AI, machine learning, computer vision, natural language processing, embedded AI, and hardware-aware AI. We select papers based on citation count, influence, and relevance to the proposed taxonomy. For recent or underexplored composition patterns, we include arXiv preprints when they present a distinct strategy or useful empirical evidence. The main collection consists of papers that clearly combine two or more lightweight methods, analyze ordering or optimization dependencies, or discuss composition-level

ablations. Single-method papers are mainly referenced as background when they clarify the role of an individual component.

Building on this scope, the main contributions of this survey are as follows:

- We comprehensively review compositions of lightweight methods for deep neural networks.
- We propose a taxonomy based on how the component methods interact: sequential application, joint optimization, and hybrid composition.
- We use comparative qualitative and quantitative analyses of prior studies to assess how method composition affects model efficiency and effectiveness.
- We identify promising yet underexplored compositions, highlight potentially incompatible methods, and provide empirical evidence to support complementary lightweight composition strategies.

The remainder of this paper is organized as follows. Section 2 reviews the principal lightweight methods and their main trade-offs. Section 3 presents the taxonomy of sequential, joint, and hybrid compositions, including those for large language models. Section 4 synthesizes quantitative evidence on the accuracy and efficiency of representative compositions. Section 5 examines their latency and memory behavior on physical hardware. Section 6 analyzes recurring complementary and conflicting interaction patterns and provides practical guidance for selecting, ordering, implementing, and deploying lightweight-method compositions. Section 7 identifies unresolved challenges and future research directions. Finally, Section 8 summarizes the findings and implications for efficient real-world deployment.

2 Overview of individual lightweight methods

This section reviews five main lightweight methods: architecture design, decomposition, pruning, quantization, and Knowledge Distillation (KD), and introduces the terminology and method-level trade-offs considered in the later composition analysis. Figure 1 illustrates the taxonomy of these lightweight methods.

2.1 Efficient Architectures

Early research focuses on designing efficient architectures by arranging components such as convolution, activation, and pooling layers in various orders, and/or by tuning model parameters such as width, depth, and input size Simonyan and Zisserman (2015); Szegedy et al. (2015); Huang et al. (2017); Han et al. (2017); Zagoruyko and Komodakis (2016); Hu et al. (2018); Tan and Le (2019). Although these hand-crafted architectures have been successful, the design process is time-consuming and slow to scale. The search space of possible architectures is vast due to many dependencies, which often hinders hand-crafted design from finding an optimal or even suboptimal architecture.

To overcome these limitations, Neural Architecture Search (NAS) has emerged in the field of AutoML Zoph and Le (2017). NAS automatically discovers efficient network architectures that achieve a favorable balance between accuracy and resource efficiency for a given application. NAS, however, inherently requires substantial computational

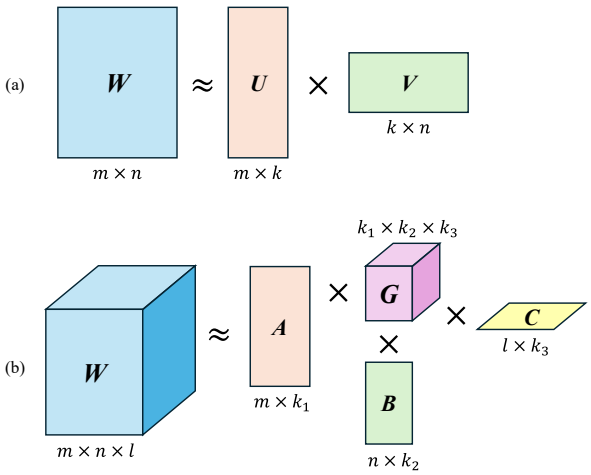


Fig. 2: (a) Low-rank matrix factorization (b) Low-rank tensor decomposition (Tucker decomposition) Liu and Parhi (2023): A matrix or tensor can be decomposed into the product of lower-dimensional components, such as smaller matrices or tensors. Decomposition reduces the number of parameters, thereby lowering both memory requirements and computational cost.

resources due to the iterations. Subsequent research efforts focus on reducing the computation required for NAS Pham et al. (2018); Liu et al. (2018, 2019), and several studies He et al. (2018); Tan et al. (2019); Wu et al. (2019); Cai et al. (2019, 2020) have used NAS to search for architectures tailored to specific applications and hardware platforms. Thus, efficient architectures are most useful when the deployed model must be small from the start. However, their effectiveness can depend strongly on the task, hardware, and search space. This motivates the later use of pruning, quantization, or KD when further compression or recovery is required.

Recently, network architectures based on transformers Vaswani (2017) have been increasingly used in various applications, including large language models (LLMs) Devlin et al. (2019); Radford and Narasimhan (2018), computer vision Dosovitskiy et al. (2021); Carion et al. (2020); Liu et al. (2021), image generation Ramesh et al. (2021); Chen et al. (2020), and speech recognition Gulati et al. (2020). Transformers, however, require extensive matrix multiplications, resulting in high computational complexity. Many studies therefore reduce attention computation by limiting attention to a subset of token pairs Kitaev et al. (2020); Beltagy et al. (2020); Li et al. (2019); Zaheer et al. (2020). Li et al. (2022) further suggest that patch-embedding and linear layers, rather than attention modules, are the main latency bottlenecks in vision transformers (ViTs) on mobile devices. For comprehensive overviews, see Liu et al. (2017); Li et al. (2021); Nan et al. (2019); Mi et al. (2022); Chen et al. (2024); Tay et al. (2020).

2.2 Decomposition

Matrix (or tensor) decomposition is a popular technique to reduce the computational load in neural networks by breaking down complex matrix multiplications, such as fully connected layers and convolutional layers, into several smaller multiplications. Because matrix multiplications account for most of the computational cost of neural networks, decomposing them significantly reduces overall complexity and accelerates inference without substantial loss in accuracy.

Tensor or matrix decomposition methods Sainath et al. (2013); Tai et al. (2015); Zhang et al. (2015,?); Shim et al. (2017); Saragadam et al. (2024); Yu et al. (2017); Hameed et al. (2022); Liebenwein et al. (2021), such as low-rank decomposition and singular value decomposition (SVD), factorize weight tensors into lower-rank components, reducing parameters and computational complexity while maintaining model performance (Figure 2). Decomposition is attractive when dense, low-rank operators can be executed efficiently. However, the rank choice controls the accuracy-efficiency trade-off, and aggressive decomposition may require KD or fine-tuning to recover performance. For more detailed discussion, see Liu and Parhi (2023); Deng et al. (2020).

Researchers have also investigated lowering the computational cost of convolution by decomposing complex operations into simpler, more efficient components. VGG Simonyan and Zisserman (2015) shows that a convolution with a large kernel, such as 7×7, can be decomposed into multiple sequential 3×3 convolutions. Similarly, the GoogLeNet studies Ioffe and Szegedy (2015); Szegedy et al. (2016, 2017) propose decomposing a $K \times K$ convolution into $K \times 1$ and $1 \times K$ convolutions. Grouped and depthwise separable convolutions follow a similar idea by reducing inter-channel computations and then restoring channel interactions with operations such as channel shuffling or pointwise convolution Ma et al. (2018); Zhang et al. (2018); Guo et al. (2018); Howard (2017); Sandler et al. (2018).

2.3 Pruning

Pruning reduces model complexity by removing unnecessary weights from a trained network, particularly weights close to zero, because they contribute little to the output. Early methods Han et al. (2015, 2016) iteratively prune and fine-tune the model to shrink its size with minimal performance loss. Pruning strategies are commonly categorized by what is removed and how importance is measured Cheng et al. (2024). Importance is typically measured using weight magnitude, gradients, or regularization that drives unimportant parameters toward zero Wen et al. (2016); Louizos et al. (2018); Scardapane et al. (2017); Liu et al. (2017).

Based on granularity, pruning can be divided into unstructured and structured pruning Unstructured pruning removes individual weights Han et al. (2015, 2016); Guo et al. (2016); Molchanov et al. (2017), but it often yields limited latency gains because sparse weight patterns make matrix multiplications irregular. Structured pruning removes entire channels or layers Li et al. (2017); He et al. (2018); Yu et al. (2018); Peng et al. (2019); Anwar et al. (2017); He et al. (2017); Liu et al. (2017);

Ye et al. (2019); Zhao et al. (2019); Kang and Han (2020), keeping the pruned model dense and more hardware-friendly.

Pruning is most effective for redundant models, but its benefit may be limited for compact models or hardware without efficient sparse kernels. Because removing weights or channels can still affect accuracy, pruning is often followed by fine-tuning or KD-based recovery. Several survey papers provide comprehensive overviews of pruning techniques Liu et al. (2020); Augasta and Kathirvalavakumar (2013); He and Xiao (2023); Vadera and Ameen (2022); Xu et al. (2020); Blalock et al. (2020).

2.4 Quantization

Quantization reduces numeric precision by mapping high-precision floating-point values to lower-precision representations, often integers. Parameters and activations are typically stored as 32-bit float and quantization encodes them with fewer bits such as 16 or 8 instead of 32, reducing the model size by half or a quarter. Beyond shrinking memory, quantization can boost throughput when both weights and activations are quantized because integer arithmetic is generally faster than floating-point on modern processors. The reduced precision, however, introduces quantization errors that can harm accuracy.

The amount of quantization error depends on the quantization intervals, each of which defines the range of original values mapped to the same quantized value. Uniform quantization uses intervals of equal width, which simplifies quantization but can significantly degrade accuracy when the original values have a highly skewed distribution. In contrast, non-uniform quantization adjusts the interval width depending on the original value distribution Cai et al. (2017); Faraone et al. (2018); Tung and Mori (2018); Frederick and Greg (2018); Yang et al. (2019); Gupta et al. (2015); Jung et al. (2019); Wu et al. (2016); Zhang et al. (2018).

Quantization methods can be divided into static quantization and dynamic quantization. Static quantization uses a predefined interval that is determined before inference and fixed during inference Jacob et al. (2018); Zhao et al. (2019). In contrast, dynamic quantization adjusts the interval based on the distribution of input values at runtime Zhang et al. (2018); Esser et al. (2020); Bhalgat et al. (2020). Quantization techniques can also be grouped into Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT), depending on when quantization is applied Banner et al. (2019); Cai et al. (2020); Choukroun et al. (2019); Fang et al. (2020); Li et al. (2021); Nagel et al. (2019); Zhao et al. (2019); Fan et al. (2021); Zhuang et al. (2018); Cai et al. (2017); Leng et al. (2018); Liu et al. (2018); Nagel et al. (2020). PTQ is easy to compose after pruning or architecture search, whereas QAT can be jointly optimized with pruning, KD, or NAS. PTQ is preferable when retraining cost is important, whereas QAT offers better adaptation to quantization error in aggressive low-bit settings at the cost of additional training. For an in-depth overview, see Gholami et al. (2022); Rokh et al. (2023); Nagel et al. (2021); Weng (2021); Guo (2018); Kulkarni et al. (2022); Mishra et al. (2020); Wei et al. (2024).

2.5 Knowledge Distillation (KD)

Unlike pruning and quantization, which reduce the size of a large model, KD builds a compact model from scratch. Training a small model entirely from scratch often results in lower performance than a large model. To address this issue, KD transfers knowledge from a large model, called the teacher, to the small model, known as a student model. This transfer helps the student model perform better than it would if trained alone.

KD methods can be grouped based on the type of knowledge transferred from the teacher model to the student model. These categories include response-based, feature-based, and relation-based KD Gou et al. (2021). In response-based KD, the student is trained on the teacher's output probability distribution, often called soft labels or dark knowledge, rather than only on one-hot ground-truth labels Romero et al. (2015); Hinton (2015); researchers have also transferred the teacher's logits directly Ba and Caruana (2014). Feature-based KD transfers intermediate features to the student Kim et al. (2018); Heo et al. (2019); Chen et al. (2021), while relation-based KD preserves relationships between data points or output features Yim et al. (2017); Tung and Mori (2019); Lee et al. (2018); Zhang and Peng (2018); Wang et al. (2020); Passban et al. (2021).

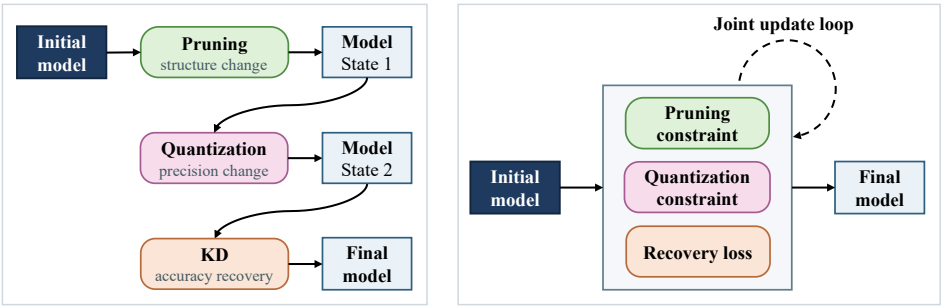
KD usually relies on a pretrained teacher model to guide the student, but knowledge transfer can also be done in parallel by training both teacher and student simultaneously, known as online KD Zhang et al. (2018); Zhu et al. (2018); Song and Chai (2018); Chen et al. (2020). A specialized form of online KD is self-distillation, in which the teacher and the student are parts of the same model Zhang et al. (2019, 2021). In compositions, KD often serves as a guidance or recovery mechanism, training better compact models or restoring accuracy after pruning, quantization, or decomposition. For broader surveys, see Gou et al. (2021); Kaleem et al. (2024); Wang and Yoon (2021).

3 Composing lightweight methods

This survey focuses on compositions because a single lightweight method may not satisfy accuracy, memory, and latency constraints simultaneously. A composition uses two or more lightweight methods in one model-development or deployment pipeline, where at least one method affects another through the model state, constraints, objective, architecture, or optimization decisions. Independent comparisons, applications to unrelated models, and alternative baselines without such dependence are excluded.

We broadly classify the composition of lightweight methods into the following three categories: *sequential application*, *joint optimization*, and *hybrid* approaches. *Sequential application* performs multiple techniques one after another, whereas *joint optimization* refines the network using multiple lightweight methods simultaneously (Figure 3). The *hybrid* approach combines sequential and joint optimization strategies. Table 1 summarizes the existing compositions of lightweight methods.

The three categories are distinguished by the optimization dependency between component methods. In sequential application, one method produces a compressed model that is subsequently used as input to another method, but earlier stages are not



(a) Sequential Application (b) Joint Optimization

Fig. 3: Comparison of sequential application and joint optimization strategies for composing lightweight methods. Pruning, quantization, and KD are shown as representative examples. (a) In sequential application, lightweight methods are applied as separate stages, so each stage inherits the model state produced by the previous one. (b) In joint optimization, multiple method-specific constraints or losses are optimized under a shared objective, allowing their interactions to be considered during the update loop.

updated using feedback from later stages. In joint optimization, multiple lightweight methods are integrated into the same training, search, or fine-tuning process. This allows their variables or objectives to be optimized with mutual influence. Hybrid methods combine these two patterns: at least one part of the pipeline is jointly optimized, while another part is introduced as a separate sequential stage.

This taxonomy captures not only which lightweight methods are combined, but also how their optimization decisions interact. Accordingly, the same pair of methods may be assigned to different categories depending on whether they are applied as independent stages, optimized under a shared objective, or integrated into a multi-stage pipeline. For instance, pruning and quantization are sequential when quantization follows pruning, but joint when pruning masks and quantization parameters are optimized together. Implementation-level variants, such as alternating and simultaneous updates in joint optimization, are discussed as subpatterns within each category rather than treated as separate top-level classes.

Table 1: Overview of surveyed compositions of lightweight methods. Strategy: Seq.=sequential, Joint=joint optimization, Hybrid=mixed. Methods: Ar=architecture/NAS, De=decomposition/low-rank, Pr=pruning, Qt=quantization, KD=knowledge distillation, PEFT=parameter-efficient fine-tuning (LoRA), AO=activation optimization, MoE=mixture-of-experts routing.

Ref.	Strat.	Meth.	Task	Model	Main conclusion
Han et al. (2016)	Seq.	Pr+Qt	Img. cls.	CNN	Prune, quantize, then Huffman-code for 35–49× compression.

Ref.	Strat.	Meth.	Task	Model	Main conclusion
Hawks et al. (2021)	Seq.	Qt+Pr	Low-lat.	Compact	QAT before pruning yields sparser low-latency models.
Zhang et al. (2022)	Seq.	Pr+Qt	Img. cls.	CNN	Channel pruning then QAT keeps accuracy at high compression.
Choi et al. (2020)	Seq.	Pr+Qt	Img. cls.	CNN	Learnable pruning/quantization regularizers close the precision gap.
Kim and Rush (2016)	Seq.	KD+Pr	NLP (MT)	Seq2seq	Sequence-level KD lets later pruning double compression.
Aghli and Ribeiro (2021)	Seq.	Pr+KD	Img. cls.	CNN	A pruned teacher avoids transferring redundant information.
Prakosa et al. (2021)	Seq.	Pr+KD	Img. cls.	CNN	KD after pruning recovers pruning-induced accuracy loss.
Wu et al. (2020)	Seq.	Pr+KD	Img. cls.	CNN	KD improves a pruned model as a post-processing step.
Chen et al. (2021)	Seq.	Pr+KD	Img. cls.	CNN	Distilling from the original restores a better pruned model.
Xie et al. (2021)	Seq.	Pr+KD	Re-ID	CNN	Pruning with KD efficiently compresses re-identification models.
Hua et al. (2019)	Seq.	Pr+KD	Img. cls.	CNN	Dynamic channel gating skips computation while KD keeps accuracy.
Liu et al. (2021)	Seq.	Pr+KD	Img. synth.	GAN	KD-guided content-aware pruning compresses GANs.
Mishra and Marr (2017)	Seq.	Qt+KD	Img. cls.	CNN	Apprentice distills a full-precision teacher into a low-precision net.
Yuan et al. (2021)	Seq.	Qt+KD	Img. cls.	CNN	KD enables low-bit quantization with limited training data.
Wu et al. (2020)	Seq.	Pr+Qt+KD	Img. cls.	Binary	SBNN slims binary nets via pruning, binarization, and KD.
Dong and Yang (2019)	Seq.	Ar+KD	Img. cls.	CNN	Transformable search prunes width/depth, then KD transfers knowledge.
Ma et al. (2023)	Seq.	Pr+PEFT	Lang. model.	LLM	LLM-Pruner structurally prunes and recovers with LoRA.
Muralidharan et al. (2024)	Seq.	Pr+KD	Lang. model.	LLM	Minitron prunes then KD-retrains with far fewer tokens.
Yao et al. (2022)	Seq.	Qt+KD	Lang. model.	LLM	ZeroQuant combines PTQ with layer-wise KD for low-bit inference.
Dettmers et al. (2023)	Seq.	Qt+PEFT	Lang. model.	LLM	QLoRA trains LoRA over a frozen 4-bit backbone to save memory.
Mozaffari et al. (2025)	Seq.	Qt+Pr+De	Lang. model.	LLM	SLiM does one-shot quantization, pruning, and low-rank correction.
Liebenwein et al. (2021)	Joint	Ar+De	Img. cls.	CNN	Finds the optimal per-layer low-rank decomposition automatically.
Li et al. (2021)	Joint	Ar+Pr	Img. cls.	Mobile	NPAS jointly does compiler-aware pruning and NAS for mobile.

Ref.	Strat.	Meth.	Task	Model	Main conclusion
Liu et al. (2019)	Joint	Ar+Pr	Img. cls.	CNN	MetaPruning generates weights for candidate pruned architectures.
Wang et al. (2020)	Joint	Ar+Pr+Qt	Img. cls.	CNN	APQ jointly searches architecture, pruning, and quantization.
Chen et al. (2018)	Joint	Ar+Qt	Img. cls.	CNN	Joint NAS and quantization finds quantization-friendly nets.
Shen et al. (2021)	Joint	Ar+Qt	Img. cls.	CNN	One-shot QAT reuses low-bit evaluations for very low-bit search.
Sheng et al. (2018)	Joint	Ar+Qt	Img. cls.	Mobile	Quantization-friendly separable convolutions reduce MobileNet error.
Kulkarni et al. (2021)	Joint	Ar+Qt	Vision (emb.)	Mobile	QF-MobileNet redesigns blocks for post-quantization accuracy.
Phan et al. (2020)	Joint	Ar+Qt	Img. cls.	Binary	Evolution-based search binarizes MobileNet effectively.
Shen et al. (2019)	Joint	Ar+Qt	Img. cls.	Binary	Searches for accurate binary neural architectures.
Bulat et al. (2020)	Joint	Ar+Qt	Img. cls.	Binary	BATS tailors the search space to binary networks.
Guo et al. (2020)	Joint	Ar+Qt	Img. cls.	CNN	SPOS one-shot NAS enables mixed-precision search via path sampling.
Li et al. (2020)	Joint	Ar+KD	Img. cls.	CNN	Block-wise NAS uses KD to supervise the architecture search.
Peng et al. (2020)	Joint	Ar+KD	Img. cls.	CNN	Cream of the Crop distills prioritized paths during one-shot NAS.
Liu et al. (2020)	Joint	Ar+KD	Img. cls.	CNN	Search-to-distill finds students best matching a teacher.
Macko et al. (2019)	Joint	Ar+KD	Img. cls.	CNN	Ensemble learning improves NAS-designed classifiers.
Kang et al. (2020)	Joint	Ar+KD	Img. cls.	CNN	Joint search is not always better than KD applied after search.
Yao et al. (2021)	Joint	Ar+Pr+KD	Detection	CNN	Joint-DetNAS unifies NAS, pruning, and dynamic distillation.
Hu et al. (2021)	Joint	Pr+Qt	Img. cls.	CNN	OPQ computes pruning mask and quantization once, then fine-tunes.
Tung and Mori (2018); Frederick and Greg (2018)	Joint	Pr+Qt	Img. cls.	CNN	CLIP-Q prunes and quantizes in parallel at each training step.
Yang et al. (2020)	Joint	Pr+Qt	Img. cls.	CNN	Sparsity-quantization joint learning can beat the sequential order.
Jung et al. (2019)	Joint	Pr+Qt	Img. cls.	CNN	Learning quantization intervals acts as implicit pruning.
Long et al. (2019)	Joint	Pr+Qt+De	Img. cls.	CNN	Low-rank regularization yields sparse quantized CNNs.

Ref.	Strat.	Meth.	Task	Model	Main conclusion
Ren et al. (2019); Ye et al. (2018)	Joint	Pr+Qt	Img. cls.	CNN	ADMM-NN splits joint pruning-quantization into ADMM subproblems.
Li et al. (2021)	Joint	De+Pr	Img. cls.	CNN	Jointly applies channel pruning and low-rank decomposition per layer.
Liu et al. (2023)	Joint	Pr+KD	Img. cls.	CNN	Jointly optimizes distillability and sparsability by loss ratio.
Bai et al. (2019)	Joint	Pr+KD	Img. cls. (FS)	CNN	Cross-distillation with pruning compresses under few-shot data.
Cho and Lee (2019)	Joint	Pr+KD	Embedded	CNN	Group sparsity and KD compress models for embedded sensors.
Yao et al. (2019)	Joint	Pr+KD	Segment.	CNN	A sparsity loss added to KD removes redundant weights.
Kim et al. (2019)	Joint	Qt+KD	Img. cls.	CNN	QKD couples quantization and KD in a co-studying schedule.
Polino et al. (2018)	Joint	Qt+KD	Img. cls.	CNN	Quantized distillation learns quantization points via KD loss.
Choi et al. (2020)	Joint	Qt+KD	Img. cls.	CNN	Data-free quantization uses a generator and KD, no real data.
Pham et al. (2023)	Joint	Qt+KD	Img. cls.	CNN	Multiple quantized teachers beat one full-precision teacher.
Wei et al. (2018)	Joint	Qt+KD	Detection	CNN	Quantization mimic matches quantized teacher/student feature maps.
Movva et al. (2022)	Joint	Pr+Qt+KD	NLP	Transf.	Pruning, quantization, and KD together give multiplicative scaling.
Wang et al. (2020)	Joint	Pr+Qt+KD	Img. synth.	GAN	GAN Slimming unifies KD, channel pruning, and quantization.
Liu et al. (2024)	Joint	Qt+KD	Lang. model.	LLM	LLM-QAT does data-free QAT distilling weights, activations, KV cache.
Xu et al. (2024)	Joint	Qt+PEFT	Lang. model.	LLM	QA-LoRA aligns quantization groups so adapters merge in quantized.
Saha et al. (2024)	Joint	Qt+De	Lang. model.	LLM	CALDERA compresses weights via joint low-precision and low-rank.
Zhang et al. (2024)	Joint	Pr+PEFT	Lang. model.	LLM	LoRAPrune uses LoRA gradients to guide structured pruning.
Klein et al. (2023)	Joint	Ar+Pr	Lang. model.	LLM	NAS-based structural pruning selects efficient LLM subnetworks.
Guo et al. (2024)	Joint	Pr+Qt	Lang. model.	LLM	JSQ co-designs sparsity and quantization to suppress outliers.
Qu et al. (2025)	Joint	Pr+Qt	Img. cls.	CNN/ViT	GETA builds a graph for joint structured pruning and quantization.
Zhang and Papyan (2025)	Joint	Pr+De	Vision	ViT	OATS splits weights into sparse outlier and low-rank terms.
Li et al. (2025)	Joint	Qt+De	Img. synth.	Diffusion	SVDQuant absorbs outliers into a low-rank branch for 4-bit diffusion.

Ref.	Strat.	Meth.	Task	Model	Main conclusion
Kim et al. (2023)	Joint	MoE+Qt	Lang. model.	MoE LLM	MoQE aggressively quantizes expert weights, keeping sparse routing.
Xie et al. (2024)	Joint	MoE+Pr	Lang. model.	MoE LLM	MoE-Pruner removes experts using router-weight information.
Chen et al. (2025)	Joint	Pr+Qt+AO	Lang. model.	MoE LLM	Expert pruning, mixed-precision, and activation opt. deploy DeepSeek-V3 under 128 GB.
Kim et al. (2021)	Hybrid	Pr+Qt+KD	KWS / img.	CNN	PQK builds a teacher from pruned weights, needing no pretrained one.
Li et al. (2020)	Hybrid	De+Pr+KD	Img. cls.	CNN	Group sparsity prunes and decomposes, then fine-tunes with KD.
Oguntola et al. (2018)	Hybrid	Pr+De+KD	Img. cls.	VGG	Pruning, factorization, and KD give an 85× smaller VGG.
Liu et al. (2021)	Hybrid	Pr+De+KD	NLP (GLUE)	BERT	ROSITA yields a 7.5× smaller BERT at 98.5% performance.
Li et al. (2024)	Hybrid	Qt+De +PEFT	Lang. model.	LLM	LoftQ jointly initializes quantization and a low-rank branch.
Guo et al. (2024)	Hybrid	Qt+De +PEFT	Lang. model.	LLM	LQ-LoRA splits weights into quantized plus low-rank parts.
Bercovich et al. (2025)	Hybrid	Ar+KD	Lang. model.	LLM	Puzzle uses distillation-NAS for 2.17× throughput at 98.4%.

3.1 Sequential Application

The simplest composition strategy is sequential application, in which individual lightweight methods are performed one after another in a predetermined order. This approach is straightforward and easy to implement, making it suitable for a wide range of applications. The preceding lightweight method may affect subsequent methods, subsequent methods do not systematically influence the preceding one.

3.1.1 Compression Methods on Efficient Architectures

Compression methods are commonly employed on large pretrained models. The typical strategy is to first achieve high accuracy with a large model and then reduce its size with a compression method while preserving the accuracy level. Interestingly, lightweight methods are often applied even to models with efficient architectures to achieve additional compression. Researchers often evaluate their proposed methods on lightweight models such as MobileNet Howard (2017); Sandler et al. (2018) to demonstrate that lightweight methods are orthogonal to efficient architecture design. Efficient architectures are combined sequentially with different compression methods, including pruning Guo et al. (2020); Gao et al. (2024); Yang et al. (2018); Gupta et al. (2024); Li et al. (2020); Liu et al. (2019), quantization Gong et al. (2019); Kim et al. (2021); Wang et al. (2022); Jin et al. (2022, 2020), and decomposition Hameed et al. (2022); Liebenwein et al. (2021). In these cases, careful design is required because efficient architectures are already highly optimized, and disrupting their structures can easily result in a significant loss of accuracy.

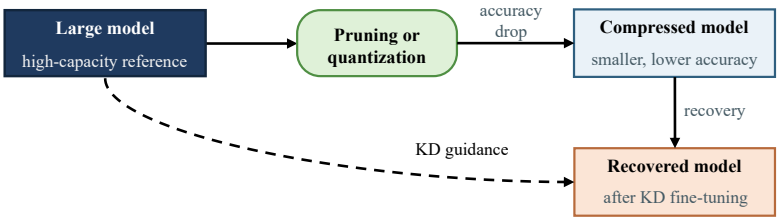


Fig. 4: Sequential compression followed by KD-based recovery. A large model is first compressed by pruning or quantization, which yields a smaller model with possible accuracy degradation. KD then transfers guidance from the original large model to fine-tune the compressed model and recover accuracy.

Efficient architecture → Pruning: The pruning technique can be easily combined sequentially with efficient architectures. Tu et al. (2020) apply multi-stage gradual pruning to depthwise separable convolution in MobileNet. The pruning algorithm can further reduce FLOPs by approximately 60% with less than a 1% loss in classification accuracy on the CIFAR-10, SVHN, and ImageNet datasets. He et al. (2018) propose to remove redundant channels in MobileNet Howard (2017); Sandler et al. (2018) using reinforcement learning. Their method reduces the FLOPs of MobileNetV1 by 60% Howard (2017) and those of MobileNetV2 by 30% Sandler et al. (2018) with an approximately 1% loss in classification accuracy on ImageNet.
Efficient architecture → Quantization: Applying quantization to efficient architectures significantly reduces computation. Baskin et al. (2021) apply quantization to MobileNet, and Sheng et al. (2018) modify MobileNet's separable convolution to apply quantization efficiently. Both methods reduce computation by more than 90% without a loss of classification accuracy on ImageNet.

3.1.2 Performance Recovery through KD

KD is often used to recover accuracy that has been degraded by the other model compression methods Mishra and Marr (2017); Wu et al. (2020); Dong and Yang (2019); Prakosa et al. (2021); Wu et al. (2020); Chen et al. (2021); Xie et al. (2021); Li et al. (2020); Hua et al. (2019); Lin et al. (2018); Blakeney et al. (2021); Yuan et al. (2021). After compression, fine-tuning with KD can recover the degraded accuracy. The uncompressed model is often used as a teacher because it likely contains more information, enabling recovery of the degraded performance by transferring undamaged knowledge to the compressed model (Figure 4).
Pruning → KD: Numerous studies on KD after pruning have demonstrated that KD consistently improves accuracy while preserving inference efficiency Dong and Yang (2019); Li et al. (2020); Wu et al. (2020); Blakeney et al. (2021); Liu et al. (2021); Prakosa et al. (2021); Wu et al. (2020); Chen et al. (2021); Xie et al. (2021); Hua et al. (2019). Li et al. (2020) present an efficient KD approach that requires only a small fraction of the original training data. They demonstrate that a student model, compressed through pruning or decomposition of the teacher model, can recover its accuracy using only 1% of the training samples. Similarly, Dong and Yang (2019)

utilize Neural Architecture Search (NAS) to identify optimal pruned architectures by searching for the best depth and width within a given model topology. The resulting pruned models are then trained using the knowledge distilled from the original model, leading to improved accuracy over simply pruned models. In another approach, Wu et al. (2020) remove insignificant channels in binary neural networks, and subsequently apply KD to further enhance model accuracy. Blakeney et al. (2021) introduce metrics designed to quantify model bias that arises after pruning, specifically focusing on scenarios where overall accuracy is maintained at the cost of lower accuracy on certain classes in classification tasks. They utilize these metrics to guide the pruning process, and subsequently employ KD loss to mitigate the induced bias. Experimental results demonstrate that their approach effectively reduces the measured bias, even when applied to imbalanced datasets.

A recent application-specific extension is CaKDP Zhang et al. (2024), which compresses LiDAR-based 3D object detectors through category-aware pruning followed by KD. The teacher's category predictions are used to measure the importance of candidate filters, so teacher information affects both the pruning criterion and the later recovery stage. Experimental result shows that a sequential pruning–KD pipeline can remain method-aware when the teacher also informs which structures should be removed.

Decomposition → KD: The accuracy loss caused by model decomposition can also be mitigated through KD. Lin et al. (2018) improve model efficiency by applying low-rank decomposition and then training the resulting lightweight model using KD. Similarly, Li et al. (2023) and Mao et al. (2020) employ KD loss to address the information loss associated with decomposition and/or pruning. Li et al. (2023) represent the weight matrix in transformer architectures as a composition of a low-rank approximation and a sparse matrix, and they train this new weight matrix with KD to enhance accuracy. Mao et al. (2020) integrate pruning with matrix decomposition. They begin with a student model initialized via distillation from a teacher model, and iteratively compress the student model to the desired size. During each iteration, the student is compressed using weight pruning and matrix factorization, followed by fine-tuning guided by the teacher model through KD. We note that these methods are designed to compress LLMs based on transformer architectures.

Quantization → KD: A similar strategy is used in conjunction with KD and quantization. Mishra and Marr (2017) propose a sequential method, where a low-precision student network is first trained independently and then further improved by distilling knowledge from a full-precision teacher network. In another work, Yuan et al. (2021) propose a post-training quantization method that works with limited data. Their approach introduces group quantization and utilizes a two-stage KD process to compensate accuracy: each network block is distilled independently, and then, all blocks are jointly fine-tuned with KD to further enhance accuracy.

3.1.3 Pruning a Knowledge-Distilled Model

Prior work has explored pruning smaller, more efficient models produced via KD. These studies argue that KD-trained models provide a strong starting point for further

optimization, as opposed to pruning already lightweight architectures. Because pruning methods are typically architecture-agnostic, applying them to a distilled student model, often based on a general-purpose architecture, can yield better results. However, since KD enhances the performance of the student model by increasing weight utilization, fewer weights or channels may need to be pruned from a student model trained with KD, compared to the original model trained from scratch. This could potentially lead to a lower compression ratio than would be achieved in a model that has not undergone KD.

KD → Pruning: Kim and Rush (2016) are the first to propose applying pruning sequentially to a knowledge-distilled model, yielding promising results. They apply pruning to a Long Short-Term Memory (LSTM) student model trained from a large LSTM teacher model to further improve the efficiency of the machine translation task. The number of parameters is reduced by 63% in the KD step with marginal performance improvement. In the pruning step, a student model with 13 times fewer parameters than the original teacher model is developed with minor performance degradation. Although the sequential composition of KD and pruning is tested solely on an LSTM model for a machine translation task, the results show that the compression contributions of KD and pruning are largely complementary.

KD with pruned teacher: Different from other studies that apply pruning on knowledge distilled model, Aghli and Ribeiro (2021); Park and No (2022) propose to use a pruned model as a teacher for KD. Aghli and Ribeiro (2021) argue that this approach helps eliminate redundancies of the teacher model, thereby preventing the transfer of unnecessary information to the student model. The method selectively prunes weights in certain layers to minimize the model architecture change when creating the teacher model. Similarly, Park and No (2022) claim that pruning the teacher model makes it more "student-friendly" and improves the effectiveness of KD. They theoretically show that the pruned teacher acts as a regularizer during distillation, thereby reducing the generalization error.

3.1.4 Quantization after Pruning / Pruning after Quantization

Pruning and quantization have been widely used to improve the efficiency of models, often with little to no loss of accuracy. Numerous studies Han et al. (2016); Choi et al. (2020); Zhang et al. (2022); Hawks et al. (2021) have investigated the sequential composition of these two techniques, which can achieve further model compression The reason for the effectiveness of combining these methods lies in their complementary nature: pruning removes less important parameters, while quantization lowers numerical precision.

Pruning → Quantization: Most studies adopting this strategy first perform pruning and then apply quantization Han et al. (2016); Zhang et al. (2022); Choi et al. (2020). This sequence is generally preferred because pruning redundant weights in the first phase enables quantization to focus more on the important weights, which reduces unnecessary computational overhead. A notable study Han et al. (2016) demonstrates a sequential approach where pruning removes redundant weights, quantization compresses the remaining weights, and Huffman coding further reduces data size, achieving significant model compression with minimal accuracy loss.

Choi et al. (2020) introduce a training framework that integrates regularization techniques for both pruning and quantization. For pruning, the method employs a partial $\mathcal{L}_2$ regularization that selectively drives weights below a certain threshold toward zero, effectively removing them. For quantization, the authors propose a mean-squared quantization error (MSQE) regularization with a learnable scaling factor, which helps to reduce the gap between high-precision weights and their low-precision approximations. Similarly, Zhang et al. (2022) adopt a two-step approach, starting with a greedy, layer-wise channel pruning guided by a data-driven importance measure. This is followed by Quantization-Aware Training (QAT) to enable efficient acceleration on hardware platforms.

Quantization → Pruning: The reverse order has also been investigated. For example, Hawks et al. (2021) propose to perform QAT first, followed by pruning on the quantized network to achieve higher compression rates. This approach allows a better understanding of quantized architecture behaviors during the pruning process, potentially leading to sparser models with improved computational efficiency compared to applying pruning or quantization alone.

3.1.5 Summary

Sequential application is beneficial when an existing model or compression pipeline with minimal changes is needed. Each technique can be added as a separate post-processing or fine-tuning stage. This approach is especially useful for situations like further compressing an efficient architecture, recovering accuracy with knowledge distillation after pruning or quantization, or combining pruning and quantization when each method addresses different types of redundancy. Its main limitations arise from the predefined method order and the large design space over method selection, ordering, composition settings, and hyperparameters. Each stage treats the output of the preceding stage as fixed, so a later method cannot revise an earlier compression decision. For example, pruning may remove information needed by subsequent quantization, in which case quantization alone may perform better than the sequential composition. Exhaustively evaluating the possible pipelines is infeasible, and limited evidence identifies the optimal order across quantization, pruning, decomposition, KD, and efficient architectures. Sequential application is therefore easy to implement but difficult to optimize globally, motivating more integrated joint or hybrid strategies.

3.2 Joint Optimization

Joint optimization integrates two or more lightweight methods within a shared training, fine-tuning, or architecture-search process. The parameters, compression decisions, or objectives associated with the component methods are optimized with mutual influence rather than as independent stages. Joint formulations may update the methods simultaneously or alternate their updates within the same optimization process.

3.2.1 Quantization and Pruning

Although quantization and pruning are often applied sequentially, as outlined in 3.1.4, numerous studies have explored approaches to jointly leverage these two lightweight techniques. To optimize a model with both lightweight methods simultaneously, studies such as Yang et al. (2020); Tung and Mori (2018); Frederick and Greg (2018); Ren et al. (2019); Ye et al. (2018) adopt an alternating strategy during training. In this approach, one method is applied to optimize the model for a fixed number of iterations while the other method remains static, and the process alternates until convergence. Despite its simplicity, this alternating optimization effectively incorporates both techniques, resulting in enhanced performance. Additionally, some studies Long et al. (2019); Hu et al. (2021); Jung et al. (2019) have proposed pruning strategies that explicitly account for quantization, thereby enabling more effective joint optimization than standard alternating optimization.

Yang et al. (2020) propose an automatic optimization of quantization and pruning to fit within the given budget, without having to manually set the compression ratio for each layer. They introduce a soft duplicate of the network weights, where the original weights are sparsified and the duplicate is quantized. These two weight copies are alternately optimized to converge toward each other. Tung and Mori (2018) propose a single learning framework that performs pruning and quantization jointly. At each training step, full-precision weights are used to determine which weights to prune via weight clipping and to compute quantization levels. The loss is then calculated using quantized weights, similar to QAT. ADMM-NN Ren et al. (2019); Ye et al. (2018) also decomposes the joint optimization problem into two subproblems and solves them iteratively using the Alternating Direction Method of Multipliers (ADMM).

Beyond alternating optimization, Long et al. (2019) aim to quantize network weights and regularize the corresponding structures simultaneously, achieving low-bit quantized networks and learning a compact structure suitable for channel pruning. They propose to apply spectral clustering regularization to the quantized network during training, which jointly optimizes quantization and regularization to encourage a compact and sparse CNN structure. Hu et al. (2021) propose to calculate the pruning mask and the quantization interval analytically only once and then use them to fine-tune the network. Although quantization and pruning do not interact with each other in this work, because the model is fine-tuned by applying two methods simultaneously, it is categorized as a kind of joint optimization. Jung et al. (2019) parameterize the quantizer and optimize the quantization interval. During the training quantization interval, this method eliminates small weight values that are negligible for quantization, which can be considered a pruning operation.

GETA Qu et al. (2025) provides a recent automatic formulation of joint structured pruning and quantization. It constructs a quantization-aware dependency graph to identify structurally coupled parameters and uses projected stochastic gradient updates to satisfy layer-wise precision constraints while pruning. Its evaluations on CNNs, BERT, and vision transformers also indicate that joint pruning–quantization frameworks increasingly need architecture-general dependency handling rather than model-specific pruning rules.

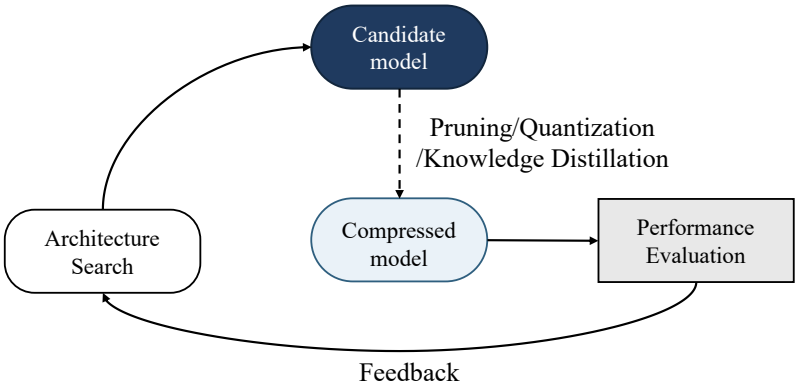


Fig. 5: Overview of compression-aware NAS: In a NAS framework, candidate models are compressed using other lightweight methods (e.g., distillation, pruning, and quantization) and evaluated to guide the next architecture search.

3.2.2 Compression-Aware Architectures:

Pruning, quantization, and KD are often applied sequentially to a chosen base network to reduce computational cost. When the architecture is fixed, the main challenge becomes developing methods that maximize compression for that particular model. However, certain architectures are naturally ill-suited to specific compression techniques, which caps the achievable gains. This motivates a complementary direction: designing or searching architectures that are inherently friendly to quantization, pruning, and KD so these methods can be applied more effectively (Figure 5).

Quantization-Aware Architectures: Sheng et al. (2018); Kulkarni et al. (2021) propose modified MobileNet architectures Howard (2017); Sandler et al. (2018) that are more suitable for quantization. Motivated by the observation that the vanilla MobileNet exhibits substantial quantization error, these works analyze the source of degradation, primarily the depthwise separable convolution blocks, and introduce architectural adjustments to improve post-quantization accuracy.

Beyond finding effective networks manually, NAS is also applied to find the optimal network architecture for a given application. Chen et al. (2018); Shen et al. (2021); Guo et al. (2020) propose to search for optimal network architectures suitable for quantization. These studies aim to achieve high efficiency by exploring models that are advantageous for quantization from scratch, as opposed to previous work that applies quantization to a given network.

Shen et al. (2021) addresses the computational overhead of NAS to find the optimal quantized model, which is mainly caused by repeated model evaluation at varying quantization levels. Instead of evaluating a quantized architecture for each target quantization level independently, this paper proposes reusing information generated when evaluating low-bit networks to efficiently evaluate higher-bit networks. SPOS Guo et al. (2020) proposes to optimize a super-network (or one-shot architecture) where each path represents an architecture candidate. Training of the super-network is performed by uniform path sampling. Thus, all architecture candidates and their weights are

trained fully and equally at once, which drastically reduces the cost of NAS model evaluation. Due to the simple and efficient optimization, SPOS supports complex search spaces such as mixed-precision quantization. This also enables the search for better lightweight architectures with different configurations, such as weight bit widths and numbers of channels. Chen et al. (2018) employ an evolutionary algorithm to discover high-performance models designed for mobile devices. In each generation, the accuracy and size of a population of quantized model candidates are evaluated. The top-performing model is mutated to generate the next generation's population, while the worst-performing model is excluded.

Binary neural networks (BNNs) are an extreme form of quantization where weights and activations are restricted to binary values, such as -1 and +1. This significantly reduces memory footprint and computational cost with risk of accuracy degeneration. Several works Phan et al. (2020); Shen et al. (2019); Bulat et al. (2020) have explored architecture search specifically tailored for BNNs to alleviate the accuracy drop caused by such aggressive quantization. Bulat et al. (2020) propose a specialized search space for BNNs that adopts the central idea of the DARTS framework Liu et al. (2019): dividing an architecture into cells and using gradient descent to find an optimal architecture for each cell. Given this narrower search space and efficient search procedure, they further optimize NAS by exploiting binary search. Shen et al. (2019); Phan et al. (2020) also propose similar methods but use evolutionary search method. Specifically, for binarizing MobileNet, Phan et al. (2020) search for optimal number of channel groups for grouped convolution in different convolution layers, and Shen et al. (2019) optimizes the network width of DoReFa-Net Zhou et al. (2016).

Pruning-Aware Architectures: Compared to studies on the design of architectures that take quantization into account, studies on optimal architectures designed for pruning appear to be relatively scarce. For quantization-aware NAS, the search space is often narrowed using quantization-bit constraints that allow efficient architecture search. Pruning-aware NAS, by contrast, requires additional training to remove unnecessary weights during architecture search, which can increase computational cost.

NPAS Li et al. (2021) proposes a layer-wise pruning approach in which the amount pruned in each layer is controlled by a pruning rate. The framework jointly performs pruning and neural architecture search, optimizing both the filter choices and the pruning strategy on a per-layer basis. In addition, it incorporates compiler-aware optimization to ensure the resulting network satisfies latency constraints on the target hardware. Klein et al. (2023) set a pretrained network as a super-network in which subnetworks represent all candidate pruned architectures. The super-network is trained once, thereby updating all candidate pruned architectures. They then perform NAS to search for pruned architectures that provide an optimal balance between accuracy and model size for LLMs. This method can be seen as a way to perform pruning, but it can also be seen as a joint optimization of pruning and architecture search from the perspective of finding an optimized architecture. Similarly, Liu et al. (2019) propose a meta-network that, given a target model, produces weights for candidate pruned architectures and uses an evolutionary algorithm to search for the best pruned

network. This search is highly efficient because the trained meta-network generates weights directly, eliminating the need for fine-tuning during search.

Beyond NAS methods that consider pruning-only or quantization-only, Wang et al. (2020) propose a unified framework that searches for architectures that are optimal for both pruning and quantization by estimating the accuracy of quantized and pruned subnetworks from a once-for-all network Cai et al. (2020) during the NAS process.

Decomposition-Aware Architectures: Searching for an optimal decomposed architecture has also been pursued. Liebenwein et al. (2021) propose an automatic layer-wise decomposition framework that jointly optimizes per-layer low-rank factors and the global compression ratio. Instead of training each decomposed network individually, the proposed algorithm automatically determines the optimal layer-wise decomposition scheme that satisfies the global compression ratio. Although formulated as a decomposition-based compression method, it can also be interpreted as a joint optimization of decomposition and architecture search, because it explores a structured, continuous space of layer configurations.

KD-Aware Architectures: The KD-aware architecture design has been actively explored Li et al. (2020); Peng et al. (2020); Liu et al. (2020); Macko et al. (2019). A central research direction is to identify student architectures that most effectively distill knowledge from the teacher network. Kang et al. (2020) apply NAS to identify an optimal student network for KD by optimizing an oracle KD loss that distills from an ensemble of teacher models. Naive NAS is computationally intensive mainly because it evaluates many candidate architectures, so other prior works introduce strategies to reduce the search cost. Li et al. (2020) propose blockwise NAS guided by distillation loss to make the NAS process more efficient. They modularize the student architecture into blocks and independently search for the optimal block architecture that can best learn knowledge from the corresponding teacher blocks. Peng et al. (2020) propose to find optimal architecture for KD in the One-Shot NAS framework where all candidate networks are presented in combinations of parts of one oracle architecture: sub-networks. One-shot NAS learns shared oracle weights at once and finds optimal sub-networks for KD. The proposed method searches for optimal sub-networks without further training, which significantly reduces the search time. Macko et al. (2019) use KD to iteratively build an ensemble of subnetworks for neural architecture search. At each step, they generate a set of small modules as weak learners guided by accuracy and distillation loss, where the teacher network is an ensemble of weak learners from the previous step. After generating sets of weak learners for all steps, they search the ensembles from all steps for the model that maximizes accuracy. Liu et al. (2020) propose to search for an optimal architecture for KD using reinforcement learning, with accuracy and efficiency after distillation on a mobile device as the reward.

3.2.3 KD with Quantization and Pruning

Just as KD can be sequentially combined with lightweight techniques, it can also be coupled with quantization and/or pruning in a joint optimization setting. In this approach, the student model is quantized or pruned while learning from the teacher, so the quantization and/or pruning objectives and the KD loss are optimized together in a single training stage.

KD with Quantization: Mishra and Marr (2017) propose both online and offline schemes for distilling from a full-precision teacher to a low-precision student. In the online setup, the teacher and student are trained jointly; that is, the student learns from the teacher, and the teacher is also updated using feedback from the student. In the offline setup, a pretrained teacher distills to the low-precision student sequentially. They report that initializing the low-precision student with weights from a full-precision counterpart yields slightly better performance than random initialization. Polino et al. (2018); Kim et al. (2019) also propose distilling knowledge from a full-precision teacher network to recover the loss of accuracy due to quantization. Polino et al. (2018) propose two joint methods. The first method incorporates distillation loss while training a smaller, quantized student network, similar to prior work Mishra and Marr (2017). The second method optimizes the location of quantization points through stochastic gradient descent to better fit the behavior of the teacher model. Kim et al. (2019) introduce a more sophisticated three-phase process. The student network is quantized without the teacher model in the first phase. The second phase is the online KD, where the teacher and the quantized student model are trained simultaneously by exchanging knowledge from each network. The last phase is the offline KD, where the teacher model is fixed and only the student model receives knowledge from the teacher model. The second phase can be viewed as joint optimization, since the teacher is also trained with the knowledge transfer to the quantized network in mind, becoming more quantization-friendly and more effective at distilling knowledge to the quantized student.

Rather than using a full-precision teacher, Wei et al. (2018) first quantize the teacher and then train a small, quantized student to learn from this quantized teacher, assuming that quantization narrows the gap between the two models and helps the student better match the teacher's feature maps. Extending this idea, Pham et al. (2023) employ multiple teachers at different quantization levels. These teachers share intermediate-layer knowledge and distill it to a low-precision student while all models are trained simultaneously, encouraging interaction between the teachers and the student. They show that using multiple quantized teachers outperforms relying on a single full-precision teacher.

Unlike other studies that use KD to improve the performance of the low-precision student model, Choi et al. (2020) exploit KD to create a quantized network in a data-free configuration. In more detail, a generator is trained to generate synthetic data similar to the original distribution using the teacher model. The synthetic data from the generator is then injected into the teacher and the student model for quantization by mimicking the output distribution of the teacher model.

KD with Pruning: Pruning can be easily integrated with KD by jointly optimizing the loss functions from both compression methods Yao et al. (2019); Bai et al. (2019); Cho and Lee (2019). Yao et al. (2019) propose to add up sparsity loss in KD process to remove redundant weights for semantic segmentation. Bai et al. (2019) introduce a cross distillation strategy in few-shot settings that can be composed with pruning to enhance the performance of compressed models using limited data. They observe slightly better results with KD compared to without it, even with few-shot training

samples. Cho and Lee (2019) apply KD and group sparsity regularization loss simultaneously to efficiently compress and deploy models on embedded systems. Liu et al. (2023) propose to jointly optimize a student network with KD and pruning compression methods with three losses: knowledge distillability, sparsability and classification loss. The ratio of knowledge distillability and sparsability losses are controlled by Dean network which is also tuned while optimizing the student network. The proposed method presents better accuracy with the same computations compared with prior works, both solely using KD or pruning and composition of KD and pruning Cho and Lee (2019); Bai et al. (2019) on image classification task.

3.2.4 Composing Three or More Lightweight Methods

So far, we have examined joint optimization of two methods. In this section, we investigate studies that integrate three or more methods.

KD + Quantization + Prune: Movva et al. (2022) examine compositions of pruning, quantization, and KD for NLP models, jointly optimizing each configuration during training. They find that using all three methods yields the best performance. Among two-method setups, KD with quantization offers the highest efficiency, while KD with pruning is the least efficient. In the field of generative model, Wang et al. (2020) propose a unified optimization framework for compressing GANs. Their approach integrates KD, channel pruning, and quantization into a single joint training process. A high-capacity generator serves as the teacher, transferring knowledge to a pruned and quantized student so the compressed model preserves image quality comparable to the original.

NAS + Pruning + KD: Yao et al. (2021) present Joint-DetNAS, a unified framework that integrates NAS, pruning, and KD for optimizing object recognition models. During architecture search, the student architecture is flexibly modified by pruning, adding, or rearranging components, and dynamic distillation is applied to find the optimal matching teachers. Joint optimization of the multiple compression methods improves the accuracy and latency of object detection models.

Pruning + Decomposition + KD: Oguntola et al. (2018) compose weight pruning, low-rank factorization, and KD on a single convolutional network and analyze the different pairwise and three-way combinations. They observe that although each method is individually effective, composing them yields multiplicative rather than additive compression, because each technique removes a different form of redundancy. Combining pruning with KD produces a VGG19 model 85× smaller on CIFAR-10 while retaining 96% of the original accuracy, illustrating that the benefit of composing three or more methods comes from their complementary redundancy targets rather than from any single technique.

3.2.5 Other Joint Compressions

Li et al. (2021) propose a collaborative decomposition and pruning approach. For each layer, they estimate how sensitive information loss is to the compression rate for each layer, then progressively remove low-importance units (channels or singular values) using a metric that consists of unit importance and the expected information loss from removing it. This composition of decomposition and pruning reduces FLOPs by

about 5% and slightly improves accuracy on CIFAR-10 compared with a pruning-only baseline.

Recent studies use low-rank decomposition to separate model information according to its sensitivity to another compression method. OATS Zhang and Papyan (2025) replaces each original weight matrix with the sum of a sparse component and a low-rank component. The sparse component stores a limited number of important individual weights, while the low-rank component compactly represents the remaining shared structure. These components are estimated together so that each captures information not represented by the other. SVDQuant Li et al. (2025) instead combines decomposition with quantization. It extracts quantization-sensitive directions into a small high-precision low-rank branch and applies four-bit quantization to the residual weights and activations. The outputs of the low-rank and low-bit branches are then combined during inference. In both methods, decomposition protects information that pruning or quantization would represent poorly, while the paired compression method reduces the remaining storage and computation.

Mixture-of-Experts (MoE) models consist of multiple relatively small expert networks and a gating module that selects the most suitable experts for each input. While MoE increases the total parameter count, inference remains efficient because only a small subset of experts is activated per example. In this perspective, MoE is considered as one of lightweight method and has composed with different lightweight methods. Kim et al. (2023) introduce Mixture of Quantized Experts (MoQE), a method that jointly leverages the MoE architecture and low-bit quantization by applying aggressive quantization to expert weights while preserving sparse expert activation. Xie et al. (2024) propose pruning MoE models by leveraging router weight information to identify and remove less important weights in the expert layers.

Beyond two-method MoE compositions, Chen et al. (2025) jointly combine expert pruning, mixed-precision quantization, and activation optimization to deploy the ultra-large DeepSeek-V3 MoE on edge hardware. They argue that pruning or quantization alone cannot meet a strict memory budget without severe quality loss, and that adding activation optimization as a third component addresses the activation-memory bottleneck that weight compression leaves untouched. The collaborative framework reduces the model storage footprint from 1.3 TB to 103 GB and enables inference under a 128 GB total memory limit while preserving higher accuracy than uniform low-bit quantization.

3.2.6 Summary

Joint optimization is beneficial when the composition needs to consider interactions among methods during training or architecture search. This includes cases like quantization-aware NAS, prune-aware NAS, KD-aware NAS, or joint pruning. It is most applicable when sufficient retraining or search budget is available. The target constraints, such as model size, FLOPs, latency, and energy, should also be defined before optimization.

The main technical challenge is optimizing multiple objectives. Sparsity, quantization error, distillation loss, architecture quality, and hardware cost may not all improve

at the same time with a simple joint loss. As a result, joint optimization can be better than sequential pipelines, but its advantages depend a lot on well-designed search spaces, loss balancing, and optimization schedules.

3.3 Hybrid

When composing multiple lightweight techniques, optimization can mix joint and sequential compression processes. For example, pruning and KD can be performed together, followed by quantization. We call this a hybrid strategy which supports many variants depending on the chosen compression methods and how they are integrated.

The composition method proposed by Kim et al. (2019), discussed in 3.2.3, can be viewed as a hybrid method. They propose three phases of model compression, and the second phase is the joint optimization of quantization and KD, and the offline KD is applied sequentially in the third phase. Kim et al. (2021) also propose to integrate pruning, quantization, and KD in a hybrid manner. In phase one, the model is jointly optimized by alternating pruning and quantization process. In phase two, the pruned network is fine-tuned as a student via KD while QAT is applied. The teacher network is constructed by combining the retained important weights and previously pruned unimportant weights, enabling mutual learning without requiring a pre-trained teacher model. This work applies two joint optimization steps sequentially within a single framework.

Li et al. (2020) propose a hybrid method that uses a joint composition of pruning and low-rank decomposition, followed by KD. Learnable sparse matrices are inserted between convolutional layers to represent pruning and low-rank decomposition. The sparse matrices are learned using a modified proximal gradient solver. The new architecture acquired in the previous step is sequentially fine-tuned with distillation loss.

Integrating the same three methods for language models, Liu et al. (2021) propose ROSITA, which combines weight pruning, low-rank factorization, and KD for BERT. Rather than fixing a single pipeline, they treat the composition as a joint design problem and explore the model architecture, KD strategy, pruning frequency, and learning-rate schedule together. They find that these design choices, rather than the individual methods, determine the quality of the composed model.Experimental results highlight that hybrid strategies that mix joint and sequential steps benefit from explicitly tuning how the constituent methods interact, rather than from adding methods in isolation.

MobileNetV4 Qin et al. (2024) combines hardware-aware architecture search with KD in a hybrid pipeline. Its Universal Inverted Bottleneck and Mobile Multi-Query Attention are optimized through two-stage NAS across mobile CPUs, GPUs, DSPs, and EdgeTPUs. Offline distilled data support the search stage, after which a separate KD recipe improves the selected models. Experimental results demonstrate that KD can support both architecture selection and final model training, while the deployment objective is evaluated across multiple hardware platforms rather than by FLOPs alone.

3.3.1 Summary

Hybrid composition is beneficial when one part of the compression process benefits from joint adaptation, while another part is better treated as a later step for refinement or recovery. The examples provided show that hybrid strategies work in complex pipelines that involve three or more methods, like pruning, quantization, KD, and decomposition. This approach is especially relevant when a single sequential or combined method cannot cover the entire optimization process.

The main challenge comes from the larger design space created by mixing stage order, combined goals, teacher-student setups, and fine-tuning schedules. Since current evidence relies on a few specific pipelines, we should see hybrid methods as flexible design patterns rather than as a better overall category.

3.4 Composition of Lightweight Methods for Large Language Models

Large language models (LLMs) have expanded the composition space because their memory requirements arise not only from model weights but also from optimizer states, activations, and the key–value (KV) cache. In addition to pruning, quantization, decomposition, KD, and architecture search, LLM pipelines commonly use parameter-efficient fine-tuning (PEFT), particularly low-rank adaptation (LoRA). LoRA does not necessarily reduce the inference cost of the frozen backbone by itself. Instead, it reduces the number of trainable parameters and can provide a lightweight recovery or adaptation stage after compression. We therefore treat PEFT as an LLM-specific composition component when its parameters, initialization, or optimization explicitly interacts with another lightweight method.

3.4.1 Pruning with Recovery

Sequential recovery remains common because pretrained LLMs are expensive to retrain jointly. LLM-Pruner first removes coupled Transformer structures and then uses LoRA to recover performance Ma et al. (2023). Its optional final INT8 stage further demonstrates a pruning–PEFT–quantization pipeline. Minitron Muralidharan et al. (2024) similarly prunes depth, attention heads, MLP dimensions, and embedding channels before KD retraining, producing compact model families with far fewer retraining tokens than independent pretraining.

3.4.2 Quantization with KD and PEFT

Quantization compositions address a different constraint: retaining a large pretrained backbone while reducing memory for weights, activations, or the KV cache. ZeroQuant uses layer-wise KD to recover aggressively quantized Transformer layers Yao et al. (2022), whereas LLM-QAT applies KD during data-free QAT and jointly quantizes weights, activations, and the KV cache Liu et al. (2024). For adaptation, QLoRA freezes a 4-bit model and trains LoRA adapters Dettmers et al. (2023). QA-LoRA instead co-designs group-wise quantization and LoRA so the adapters can be merged into the quantized weights Xu et al. (2024). These approaches illustrate that the

preferable composition depends on the target: QLoRA primarily reduces fine-tuning memory, while QA-LoRA also optimizes the final quantized representation.

3.4.3 Low-Rank and Joint Compositions

Low-rank branches are also used to isolate approximation errors that are difficult to encode at low precision. LoftQ alternates quantization and truncated SVD to initialize a quantized backbone and LoRA factors before downstream fine-tuning Li et al. (2024). LQ-LoRA similarly decomposes each matrix into a quantized component and trainable low-rank factors while allocating mixed quantization configurations under a memory budget Guo et al. (2024). CALDERA targets post-training compression by jointly optimizing a quantized backbone and quantized low-rank residual Saha et al. (2024).

Other studies make the recovery or search component influence the compression decision itself. LoRAPrune uses LoRA weights and gradients to guide structured pruning while updating only a small set of parameters Zhang et al. (2024). JSQ jointly selects sparsity and quantization settings to control outliers that are harmful to low-bit representation Guo et al. (2024). SLiM combines quantization, semi-structured pruning, and low-rank error correction in a one-shot pipeline Mozaffari et al. (2025). Puzzle uses distillation-guided architecture search to assemble an efficient LLM under a hardware throughput objective Bercovich et al. (2025).

3.4.4 Summary

LLM composition is beneficial when the methods address distinct training or deployment constraints. Pruning changes the deployed architecture, quantization reduces the memory cost of retained values, KD or PEFT can recover or adapt compressed models, and low-rank branches can preserve quantization-sensitive information. These roles allow LLM pipelines to reduce deployment cost, adaptation cost, or both.

The main challenge is that these benefits should be evaluated along two related but distinct axes. Deployment compression concerns model size, latency, throughput, energy, and KV-cache memory, whereas training-efficient adaptation concerns optimizer memory, trainable parameters, and adaptation data. A method may improve one axis without improving the other. For example, LoRA greatly reduces fine-tuning cost but does not remove the frozen backbone, while structured pruning changes the deployed architecture but may require expensive recovery. Effective LLM composition therefore requires matching the interaction pattern to the target bottleneck and reporting the cost of all retained components, including adapters, low-rank branches, sparse metadata, and quantization parameters.

4 Quantitative Analysis

When composing lightweight methods, the first question is whether the composition performs better than using each method on its own. However, fully controlled comparison across papers is infeasible because the reported results are obtained from different models, datasets, hardware platforms, and software implementations. Therefore, this section does not use results from different papers as a unified benchmark. Instead, we

focus on within-paper ablations under the same experimental setting and use cross-paper results only to identify broad trends. Given these inevitable differences in the literature, we use Accuracy Drop (AD, Equation 1) and Compression Rate (CR) as common interpretation measures: AD normalizes the quality change relative to each paper's own baseline, while CR provides a single summary measure of the reported or inferable efficiency reduction. Together, these measures provide a consistent way to describe the accuracy–efficiency trade-off of each composition.

$$AD = \frac{Baseline_{acc.} - Composition_{acc.}}{Baseline_{acc.}} \times 100 \tag{1}$$

A lower AD indicates less accuracy degradation relative to that paper's own baseline. If AD is negative, the composed method achieves higher accuracy than the baseline. CR is taken directly from the cited paper when it is reported. When CR is not explicitly reported, we estimate it from the available reduction in FLOPs/MACs, parameter count, model size, or bit width. If only the quantization level is provided, we estimate the bit-width-based CR from the ratio to FP32 precision; for example, W8A8 is treated as approximately 16× compression relative to FP32 because both weights and activations are reduced by 4× (32/8). Because some tables include studies with different CR bases, the source-specific basis is identified in the table notes rather than treated as a directly comparable quantity across all rows.

4.1 Sequential: Compression Methods on Efficient Architectures

When additional compression techniques are applied to models with efficient architectures, the potential for further computational reduction is inherently limited due to the pre-existing optimizations of these architectures. This has led to an ongoing debate about the most effective approach: whether to achieve high compression on a larger model or lower compression on a model with an efficient architecture. For pruning, the reviewed studies show that efficient architectures may achieve a smaller relative compression ratio than large models, but the final compressed model can still be more efficient in absolute terms. He et al. (2018) apply their pruning method to VGG16, achieving an 80% reduction in FLOPs, which is a higher CR compared to the same algorithm applied to MobileNet (60% for V1 and 30% for V2). Despite these differences in CR, the classification accuracy on the ImageNet dataset remains similar for both architectures. Given that MobileNetV2 has 314 million FLOPs versus VGG16's 15 billion FLOPs, simple calculations reveal that applying compression methods to more efficient architectures enables greater computational efficiency while maintaining the same level of accuracy. Similar evidence is reported by Yang et al. (2018), who show that MobileNetV1 and MobileNetV2 can be further compressed with up to a 1.7× measured latency speedup while maintaining or improving ImageNet accuracy.

This observation holds true when using quantization methods as well. When Baskin et al. (2021) apply quantization to the relatively heavy and less optimized ResNet-18, they achieve a 95% reduction in computational requirements with a 2% accuracy loss. In contrast, applying the same quantization technique to MobileNetV1 yields a 90% reduction in computation with image classification accuracy similar to that of

the quantized ResNet-18. Although quantizing ResNet-18 achieves a higher CR, the resulting model still requires 2.5 times more computation than quantized MobileNetV1 at equivalent accuracy. A similar pattern appears when quantization is combined with pruning on efficient architectures: Choi et al. (2020) apply pruning–quantization–entropy coding to MobileNet and ShuffleNet on ImageNet, achieving 7.13× and 6.79× compression ratios, respectively, with limited accuracy loss. Across these cases, large models show higher relative compression, whereas compressed efficient architectures often achieve higher deployment efficiency at comparable accuracy. However, this area has not been thoroughly explored, and further research is necessary to identify the optimal sequential combinations of models and compression techniques for various applications.

4.2 Sequential: Effectiveness of Composing with KD

KD is widely used to mitigate accuracy loss from earlier compression steps. Prior studies Prakosa et al. (2021); Chen et al. (2021); Xie et al. (2021); Liu et al. (2021) report that KD can recover part of the pruning-induced accuracy loss. As summarized in Table 2, applying KD afterward consistently and considerably reduces AD. Mishra and Marr Mishra and Marr (2017) and Wu et al. Wu et al. (2020) also demonstrate that KD can recover quantization-related accuracy loss across multiple experiments. Quantization compresses image classification models by 16× or 58×, and using KD as a subsequent step further reduces AD. Even when a model is initially jointly optimized with quantization and KD, a subsequent standalone KD stage still provides marginal gains Kim et al. (2021). These findings indicate that post-compression KD is effective on its own and serves as a flexible tool for improving the accuracy of lightweight models.

Alternatively, KD can be applied first to concentrate information in the model before subsequent pruning Kim and Rush (2016). Pruning the distilled model by 50% incurs no additional accuracy loss, while CR doubles from 2.6 to 5.3 in an NLP task. These experimental results support KD-before-pruning as a viable strategy. However, there is no clear evidence that KD-before-pruning outperforms KD-after-pruning or vice versa, highlighting the need for further study.

4.3 Sequential: Pruning → Quantization

In this section, we quantitatively examine the effectiveness of sequentially applying pruning and quantization. Table 3 reports ImageNet results for sequential pruning → quantization composition Han et al. (2016); Zhang et al. (2022); Choi et al. (2020) across different base models. In most cases, AD stays near zero, indicating that adding quantization after pruning preserves the accuracy of the pruned model while delivering additional compression.

Notably, in Zhang et al. (2022)'s experiments with DenseNet121, AD remains near zero even at $CR = 64$, showing that quantization is complementary to pruning and adds largely orthogonal compression with minimal accuracy loss. Their results further show that, for the same overall CR, pruning typically causes greater AD than quantization. For both DenseNet121 and MobileNetV2, assigning more compression

Table 2: Effectiveness of KD, sequentially combined with pruning and quantization across different architectures and datasets (CR=Compression Rate, AD=Accuracy Degradation, Pr=Pruning, KD=Knowledge Distillation, Qt=Quantization): The higher CR and the lower AD, the better. Because the rows come from different configurations, the table should be interpreted as trend evidence rather than a fully controlled benchmark comparison. The results suggest that applying KD sequentially after pruning and/or quantization consistently improves accuracy across diverse base models and datasets. When KD is applied before pruning, accuracy can be preserved while achieving a higher CR.

Method	Base Model / Dataset	$CR\uparrow$	$AD(\%)\downarrow$	
			Pr Only	Pr→KD
Prakosa et al. (2021)	MobileNets / Caltech-256	1.2	2.8	0.6
Chen et al. (2021)	ResNet32x4 / CIFAR100	10	2.7	1.1
Xie et al. (2021)	ResNet50 / Market1501	8.4	7.2	0.7
Liu et al. (2021)*	StyleGAN2 / FFHQ	23.7	3.6	3.1
			Qt Only	Qt→KD
Mishra and Marr (2017)	ResNet / ImageNet	16	3.0	1.2
Wu et al. (2020)	CNN / CIFAR10	58	-1.0	-2.2
			Qt+KD	Qt+KD→KD
Kim et al. (2021)	ResNet-50 / ImageNet	64	-0.8	-1.3
		114	0.3	-0.1
		256	3.4	3.1
			KD Only	KD→Pr
Kim and Rush (2016)	LSTM/ WMT2014	2.6	2.5	-
		5.3	-	2.5 (50% prune)
		13	-	3.5 (80% prune)

* Accuracy is measured by PSNR.
(Basis of CR) Parameter size: Prakosa et al. (2021), Chen et al. (2021), Xie et al. (2021), Mishra and Marr (2017), and Kim and Rush (2016); FLOPs/MACs: Liu et al. (2021); bit width: Wu et al. (2020) and Kim et al. (2021).

to pruning increases AD. Specifically, at $CR = 64$, 4× pruning and 16× quantization yields higher AD than 2× pruning and 32× quantization. This suggests that to

Table 3: ImageNet classification results of pruning → quantization sequential composition across different base models: Note that $CR = 1$ indicates that no corresponding compression is applied. The results consistently indicate that quantization complements pruning, providing an orthogonal compression direction with minimal loss in accuracy.

Base Model	$CR \uparrow$			$AD(\%) \downarrow$
	Prune	Quantization	All	
		Han et al. (2016)		
AlexNet	9	1	9	-0.02
	9	3	27	0
		Zhang et al. (2022)		
DenseNet121	1	16	16	-0.43
	2	16	32	-0.39
	4	16	64	3.21
	2	32	64	0.37
MobileNetV2	1	16	16	0.07
	2	16	32	1.07
	4	16	64	10.67
	2	32	64	5.33
		Choi et al. (2020)		
MobileNetV2	2	1	2	0.99
	2	16	32	0.14
	2	32	64	1.69
	2	64	128	5.64
ShuffleNet	2	1	2	0.15
	2	16	32	-0.61
	2	32	64	0.92
	2	64	128	9.02

(Basis of CR) Parameter size: Han et al. (2016) pruning/quantization, Zhang et al. (2022) pruning, and Choi et al. (2020) pruning; bit width: Zhang et al. (2022) quantization and Choi et al. (2020) quantization.

achieve a high compression ratio, the optimal balance between pruning and quantization should be explored. AD also depends on the base model. Lightweight architectures (e.g., MobileNet, ShuffleNet) generally have less redundancy Choi et al. (2020). Thus, even at relatively low CR, applying additional quantization compression after pruning leads to larger accuracy drops than in heavier models such as DenseNet121.

Table 4: Experimental comparison of quantization-aware NAS and independent quantization on ImageNet (Qt=Quantization, NAS=Neural Architecture Search): Quantization-aware NAS consistently presents better AD with higher CR than quantization, which indicates that there is an optimal architecture for quantization compression. The CR values are either reported by the original papers or estimated from stated bit-width and compression settings, and the quantization methods differ across experiments. The table should be read as source-specific trend evidence because the CR basis differs by study.

Base Model	Qt-only		Joint (Qt+NAS)	
	$CR\uparrow$	$AD(\%)\downarrow$	$CR\uparrow$	$AD(\%)\downarrow$
	Chen et al. (2018)			
ResNet50	4.1	0	6.9	-0.3
DenseNet121	4.2	0	5.2	-0.2
MobileNet-v2	4.1	5.2	3.3	-0.5
	Guo et al. (2020)			
ResNet-34, 2W2A	256	5.6	256	4.7
ResNet-34, 3W3A	114	3.3	114	1.5
ResNet-34, 4W4A	64	2.0	64	0.5
	Shen et al. (2019)			
Binary Network	3.5	10.2	3.7	1.4

(Basis of CR) Parameter size: Chen et al. (2018); bit width: Guo et al. (2020); FLOPs/MACs: Shen et al. (2019).

4.4 Joint: Compression-Aware Architecture

Quantization-Aware architecture: Sheng et al. (2018); Kulkarni et al. (2021) modify MobileNet block structures to create quantization-friendly variants, yielding 2–3% higher accuracy than a model, quantized from original MobileNet. This supports the idea that architectures can be tailored to quantization (and other compression) for better performance. Prior NAS-based studies that search for quantization-aware architectures report similar gains Chen et al. (2018); Guo et al. (2020); Shen et al. (2019). Table 4 compares the results of quantizing a fixed architecture with those of searching for an optimal architecture for quantization (Quantization-Aware NAS), and consistently shows the latter outperforming the former, reinforcing the existence of quantization-optimal architectures.

As discussed in Section 3.1, sequential pipelines can be suboptimal because changes from one method may leave the model poorly suited for the next. Evidence of this can be found in the quantization, pruning, and NAS experiments of Shen et al. (2021) and Wang et al. (2020). Fig. 4(a) of Shen et al. (2021) presents a Pareto curve comparing

Table 5: Comparison of NAS composed with KD: KD is either jointly applied during NAS or sequentially applied after NAS. The table focuses on within-paper comparisons of KD placement because the cited studies use different search spaces and training recipes. Sequential composition of NAS and KD performs better than joint composition, which is the opposite of the pattern observed for NAS and quantization. Using both joint and sequential composition marginally increases accuracy.

Dataset	Apply KD		$AD(\%)\downarrow$	Accuracy(%) ↑
	On NAS	After NAS		
		Peng et al. (2020)		
ImageNet	✗	✗	-	76.5
	✓	✗	-1.8	**77.9**
		Liu et al. (2020)		
ImageNet	✗	✗	-	73.0
	✗	✓	-2.3	<u>74.7</u>
	✓	✗	-0.8	73.6
	✓	✓	-3.4	**75.5**
		Kang et al. (2020)		
CIFAR-100	✗	✗	-	74.55
	✗	✓	-3.1	<u>76.85</u>
	✓	✗	0	74.56
	✓	✓	-3.2	**76.97**

quantization-aware NAS and NAS-then-quantize approaches under 3-bit quantization. Joint optimization achieves higher accuracy across a wide range of FLOPs, reported in millions, with at least a 2–3% improvement over independent quantization. Fig. 6 of Wang et al. (2020) compares joint NAS–pruning–quantization with a sequential approach (NAS → AMC He et al. (2018) → HAQ Wang et al. (2019)). The joint approach achieves higher accuracy for a given latency and delivers 2.3% higher accuracy under the same latency and energy constraints.

KD-Aware NAS: Table 5 summarizes experiments with KD-guided NAS across multiple datasets Peng et al. (2020); Liu et al. (2020); Kang et al. (2020). Depending on when KD is applied, we distinguish two settings: (1) joint, where KD is used during the architecture search process, and (2) sequential, where KD is applied after the architecture is found. As shown in Table 5, architectures discovered with KD generally achieve higher accuracy than those searched without KD. Notably, Liu et al. (2020); Kang et al. (2020) report that applying KD after the search outperforms applying KD during architecture search, indicating that joint optimization is not always superior to sequential lightweight method composition. As expected, using KD in both stages yields the best results, likely because KD-guided architectures are inherently more compatible with KD, enabling better knowledge transfer and improved performance.

Prune-Aware NAS: Prune-aware NAS has shown promising results. Li et al. (2021) demonstrate that explicitly accounting for pruning during the search yields architectures with higher accuracy than those produced by conventional NAS that ignores pruning. On the ImageNet classification benchmark, the prune-aware NAS approach achieves $AD = 4.1$ while providing roughly twice the CR of NAS alone.

4.5 Joint: Effectiveness of Composing with KD

KD with Pruning: In comparing the sequential application of pruning and KD with joint optimization, Liu et al. (2023) investigate KD and pruning individually, as well as their sequential and joint compositions. The findings reveal that KD alone improves accuracy by 1.25% over the baseline on the CIFAR-10 dataset, while pruning alone achieves a smaller increase of 0.89%. When KD and pruning are applied sequentially, the accuracy improvement is 1.14%, indicating limited synergy between the two methods when combined in sequential format. In contrast, the joint optimization approach leads to the highest accuracy improvement of 1.94%, showcasing the better effectiveness of the joint optimization method than sequential one.
KD with Quantization: Similar to the results observed with sequential optimization of KD with quantization, the joint optimization of quantization and KD also achieves a better result, compared to the individual application of quantization Choi et al. (2020). However, joint optimization of KD and Quantization sometimes presents different results from those observed in sequential compression. Researchers Mishra and Marr (2017); Pham et al. (2023) show that joint optimization offers no significant advantage over the sequential method in certain scenarios. In Mishra and Marr (2017), the joint training of teacher and low-precision student model results in slightly lower performance than the independent application of KD to the low-precision student. Pham et al. (2023) also report that the naive joint training does not offer better advantage than individual KD. After applying their proposed method, accuracy is improved by $2-3\%$. This suggests that joint optimization by itself does not necessarily yield substantial performance gains. Appropriate approach is required that suits the characteristics of each method, rather than relying on naive joint optimization for better performance. In practice, the performance impact depends heavily on how the optimization procedure is designed and implemented.

4.6 Joint: Quantization and Pruning

Quantization and pruning target complementary aspects of model redundancy; therefore, their sequential composition raises the compression ratio CR while keeping the AD low, as presented in Section 4.3. When they are composed jointly, similar behaviors are also observed in prior work Yang et al. (2020); Long et al. (2019). The results from Yang et al. (2020) in Table 6 show that jointly applying both methods achieves a significantly higher CR with an on-par AD compared with applying either quantization or pruning alone. This result is from the CIFAR dataset, which is highly overparameterized with ResNet-50 (a large model on a relatively small dataset of 60K images). In particular, when pruning and quantization are jointly applied to the relatively large ResNet-50 model, CR can reach 462 while accuracy is maintained, whereas

Table 6: Image classification performance between joint vs individual applications of pruning and quantization: Within each cited study, jointly optimizing a model with both pruning and quantization generally presents higher CR and comparable AD than compressing a model with a single lightweight method. Because the table includes different datasets and base models, comparisons across rows should be interpreted with caution.

Base Model	Dataset	Prune	Quant.	$CR \uparrow$	$AD(\%) \downarrow$
		Yang et al. (2020)			
ResNet-20	CIFAR	✗	✓	16	0
		Joint		35.4	0.15
ResNet-50	CIFAR	✓	✗	2	-1.61
		Joint		462	-1.34
AlexNet	ImageNet	Joint		205	-0.14
		Long et al. (2019)			
ResNet-18	CIFAR	✗	✓	1.1	-0.39
		Joint		1.3	-0.35

(Basis of CR) Parameter size: Yang et al. (2020); FLOPs/MACs: Long et al. (2019).

pruning alone compresses the model by a factor of two. By contrast, Long et al. (2019) report mixed results in Table 6. The combined approach does not clearly outperform quantization-only. A plausible reason is that their base model, ResNet-18, is already compact, so the compression may be near saturation.

A natural question is whether quantization and pruning should be applied sequentially or jointly. Evidence from Yang et al. (2020) indicates that the joint approach can outperform the sequential one on AlexNet when evaluated on ImageNet. Compared with the sequential AlexNet result in Han et al. (2016) (Table 3), the joint AlexNet result reports better accuracy and a larger parameter-size CR, but this cross-paper comparison should be interpreted cautiously, as it is not controlled benchmark comparison. While more effective sequential strategies may exist, the joint approach is generally expected to yield superior performance.

4.7 Hybrid

Hybrid approaches can be useful when a jointly optimized compression stage is followed by a separate refinement stage to recover performance or improve the compressed model. PQK Kim et al. (2021) shows this by first combining iterative pruning with QAT and then refining the pruned network through KD. As shown in Table 2, this second hybrid phase improves AD from -0.8 to -1.3 at $CR = 64$, from 0.3 to -0.1 at $CR = 114$, and from 3.4 to 3.1 at $CR = 256$ on ResNet-50/ImageNet. This suggests that a recovery stage can still provide measurable gains even after pruning and quantization have already been optimized together.

The same pattern appears in hybrid pruning–decomposition methods. Group sparsity Li et al. (2020) learns a compact model that can act as filter pruning or low-rank decomposition. The authors report that using the distillation loss further reduces the Top-1 error rate after compression. In ROSITA Liu et al. (2021), adding KD-guided depth pruning to KD-guided width pruning raises the average development score from 79.0 to 79.4. These results suggest that hybrid composition is most effective when the jointly optimized stage and the later refinement stage have distinct roles, rather than using multiple methods redundantly.

4.8 Others

For the image generation task, Wang et al. (2020) investigate various compositions of the three methods. Applying pruning followed by KD resulted in slightly better performance than pruning alone, whereas reversing the order led to worse results. Notably, the joint optimization of KD and pruning achieves significantly improved performance. Furthermore, when incorporating quantization, the joint optimization of all three methods (KD, pruning, and quantization) outperforms the approach of applying quantization after the joint optimization of KD and pruning.

Beyond vision tasks, Movva et al. (2022) apply quantization, pruning, and KD to compress the BERT architecture, demonstrating the effectiveness of joint optimization in NLP tasks. They evaluate the three methods individually and in several compositions, including KD with QAT, QAT with magnitude pruning (MP), KD with MP, and the combination of QAT, KD, and MP. In their experiments, the application of individual MP results in a slight performance improvement. Meanwhile, the composition of KD and quantization leads to a notable increase of approximately 2–3% in the BLEU score for BERT-Mini. The best performance is achieved when all three methods are applied together, resulting in an improvement of over 5%.

5 Real-Hardware Deployment Evaluation

Section 4 uses accuracy and analytical efficiency measures, including parameter reduction, FLOPs, bit width, and estimated compression rate, to compare method compositions. These measures describe model-level compression but do not directly predict deployment performance. Parameter count and bit width approximate model storage. Floating-point operations (FLOPs) count arithmetic operations, while bit operations (BOPs) also account for operand precision. Neither measure captures runtime memory for activations, temporary buffers, sparse indices, quantization metadata, and low-rank branches. Latency additionally depends on memory movement, kernel support, parallelism, compiler optimization, and the target processor. Consequently, a composition with a larger theoretical compression rate may be slower on physical hardware Ma et al. (2023); Li et al. (2025); Qin et al. (2024).

Table 7 summarizes representative composition studies that report measurements on physical hardware. The results are not intended as a controlled cross-study comparison because the models, tasks, devices, runtimes, precisions, batch sizes, and input lengths differ. Instead, Table 7 shows which deployment effects become visible only when the complete composition is executed on its target platform.

The Table 7 highlights three deployment considerations. First, the composition should target the resource that most limits deployment, such as memory capacity or inference latency. Quantization can reduce memory even when latency does not improve, as shown by LLM-Pruner Ma et al. (2023), while structured pruning, dense decomposition, and hardware-aware architecture search can preserve operators that existing runtimes execute efficiently Li et al. (2021); Lin et al. (2018); Qin et al. (2024). Sparse models provide wall-clock acceleration only when the target library or accelerator contains a matching sparse kernel Mozaffari et al. (2025). Second, evaluation of the complete pipeline reveals costs hidden by layer-level or theoretical measurements. Sparse metadata, low-rank recovery, quantization and dequantization, format conversion, and device transfer can offset arithmetic savings. SVDQuant demonstrates the effect of kernel fusion Li et al. (2025), while the layer-wise GPU speedups reported by SLiM Mozaffari et al. (2025) do not establish complete generation speedup. Third, results remain platform dependent because memory hierarchy, numerical support, compiler transformations, and kernel availability vary across CPUs, GPUs, DSPs, and accelerators Qin et al. (2024).

Overall, deployment evidence should be judged by how directly it reflects the target execution setting. Physical-device measurements provide the strongest evidence because they capture the combined effects of hardware, software stack, precision, workload, and runtime behavior.

6 Practical Guidance for Composing Lightweight Methods

This Section translates the interaction and deployment evidence into practical guidance for selecting, ordering, implementing, and evaluating lightweight-method compositions. The appropriate composition depends on many constraints, the recommendations are conditional rather than universal Qu et al. (2025); Jeon et al. (2025); Li et al. (2025). For practical guidance, Sections 6.1 and 6.2 explain when methods provide complementary benefits and when their interactions create conflicts. Section 6.3 then summarizes libraries for creating, executing, and serving composed models.

6.1 Complementary Compositions

Lightweight methods are complementary when they reduce different costs, repair a specific error introduced by another method, or co-adapt decisions that cannot be optimized independently Han et al. (2016); Yao et al. (2022); Frantar and Alistarh (2022). Compatibility depends on compression severity, order, optimization coupling, and the deployed representation rather than on the method alone Guo et al. (2024); Jeon et al. (2025).

Assign each method a distinct role. Pruning and quantization can accumulate savings because pruning removes structures and quantization lowers the precision of retained values Han et al. (2016); Zhang et al. (2022). A sequential prune-then-quantize pipeline is a practical starting point at moderate compression. When sparsity and precision act on the same weights at aggressive settings, use quantization-aware pruning or a shared reconstruction objective because retained structures become more

Table 7: Representative real-hardware evaluations of lightweight-method compositions. The arrow denotes sequential and + denotes joint composition. MobileNetV4 is hybrid. Note that results are not directly comparable because hardware, software, and workloads differ.

Study / composition	Measured model and physical setup	Measured deployment result	Deployment observation
MobileNetV4 Qin et al. (2024): KD-guided NAS → KD	ImageNet-1K. Median latency on Pixel 6 CPU, Pixel 8 EdgeTPU, iPhone 13, Pixel 4 DSP, Pixel 7 GPU, and Samsung S23 CPU/GPU.	Hybrid-L obtains 83.4% top-1 at 3.8 ms on Pixel 8 EdgeTPU. The KD recipe raises accuracy to 86.6% without changing the architecture.	Architecture search targets latency, while KD improves accuracy. Processor-dependent operator efficiency motivates evaluation across the intended devices.
NPAS Li et al. (2021): NAS + pruning	ImageNet classification on an off-the-shelf mobile phone with compiler-generated inference code.	Representative operating points are 78.2% top-1 at 6.7 ms, 75.0% at 5.9 ms, and 71.0% at 3.9 ms.	Joint selection of pruning patterns, architecture, and compiler configuration directly targets the mobile latency budget rather than FLOPs alone.
LRDKT Lin et al. (2018): decomposition → KD	AlexNet and VGG-16 on ImageNet. CPU and NVIDIA GTX Titan X GPU with GPU batch size 32.	VGG-16 LRDKT-0.5 reports 2.61× CPU speedup and 2.55× GPU speedup.	KD recovers accuracy after decomposition while the resulting dense operators retain practical CPU and GPU acceleration.
LLM-Pruner Ma et al. (2023): pruning → LoRA → INT8	LLaMA-7B and Vicuna-7B on one NVIDIA A5000. Inference uses a 64-token input.	Dense: 12,884.5 MiB and 69.32 s. Pruned 5.4B: 10,488.4 MiB and 58.55 s. Pruned+INT8: 5,444.4 MiB and 63.10 s.	INT8 nearly halves the pruned model's memory but increases latency in this implementation, exposing a memory–latency trade-off hidden by model size.
SVDQuant Li et al. (2025): low rank + W4A4	FLUX.1, PixArt-Σ, and SDXL on laptop and desktop RTX 4090 and RTX 5090 GPUs.	FLUX.1 BF16 uses 22.7 GiB and 111.7 s, versus 6.5 GiB and 12.9 s for fused INT4. An unfused low-rank branch adds 18% single-step latency.	Kernel fusion prevents the low-rank recovery branch from offsetting the acceleration obtained from four-bit execution.
ZeroQuant Yao et al. (2022): quantization → KD	BERT, GPT-J-6B, and GPT-NeoX-20B on NVIDIA A100 40 GB GPUs with fused INT8 kernels.	BERT reports up to 5.19× speedup. GPT-J latency decreases from 29.13 to 7.94 ms. GPT-NeoX deployment decreases from two GPUs to one.	Fused quantization overhead enables the measured INT8 speedup, while KD primarily supports accuracy recovery in more aggressive low-bit settings.

precision-sensitive Frantar and Alistarh (2022); Guo et al. (2024); Qu et al. (2025). Avoid assigning heavy structural compression to an architecture that already has little redundancy Zhang et al. (2022).

Use recovery only for damage it can represent. KD has recovered accuracy after pruning, decomposition, and quantization when the compact student remains

trainable and the teacher provides targets for the affected representation Lin et al. (2018); Yao et al. (2022); Muralidharan et al. (2024). Sequential recovery is easy to add to an existing pipeline, but its effectiveness depends on teacher quality, recovery data, and compression severity Mishra and Marr (2017); Li et al. (2020); Yuan et al. (2021). For LLMs, pruning followed by KD or PEFT follows the same recovery pattern, although recovery can remain incomplete after structural removal Ma et al. (2023); Muralidharan et al. (2024). Define the compact structure first, then apply KD while enough capacity remains for recovery. For low-bit LLM adaptation, a separate LoRA adapter is appropriate when four-bit adaptation memory is the main constraint, whereas quantization-aware or group-aligned PEFT is more suitable when precision is lower or the adapter must be merged Dettmers et al. (2023); Xu et al. (2024); Jeon et al. (2025).

Choose sequential or joint optimization from variable dependency. Sequential composition is sufficient when a later method can consume a fixed first-stage model without changing its masks, groups, or architecture. Joint optimization becomes useful when methods perturb the same parameters, share a compression budget, or require a common deployment representation Frantar and Alistarh (2022); Qu et al. (2025); Jeon et al. (2025). A hybrid pipeline is appropriate when tightly coupled methods require joint adaptation but another method is better reserved for later recovery or refinement Kim et al. (2019, 2021); Li et al. (2020). This flexibility increases the design space because both within-stage coupling and between-stage ordering must be selected. Before adopting a joint or hybrid formulation, compare it with both application orders at matched resources because additional coupling adds value only when the variables are genuinely dependent.

Partition by compression sensitivity. Sparse and low-rank components can represent localized outliers and distributed information separately, while low-rank branches can protect directions that low-bit quantization handles poorly Zhang and Papyan (2025); Li et al. (2025). Use this pattern only when the residual assigned to the low-rank branch is compact enough to justify its rank, memory, and additional execution. A third method is useful when it corrects the aggregate error of earlier stages or addresses a separate resource bottleneck, as in low-rank correction after pruning and quantization Mozaffari et al. (2025).

Include deployment behavior in architecture and composition search. Compression-aware NAS can avoid selecting an architecture that performs well before compression but poorly after quantization or pruning Wang et al. (2020); Qin et al. (2024); Gao et al. (2024). Use this approach when search is affordable and the target platform is known. Evaluate the complete exported model with the intended compiler and kernels rather than using FLOPs as the only objective Qin et al. (2024); Gao et al. (2024).

Validate the complete composition. Count sparse metadata, quantization scales, adapters, low-rank branches, and format conversions. Measure end-to-end quality, latency or throughput, peak memory, and energy on the target workload because a corrective branch or unsupported sparse format can remove the expected deployment benefit Li et al. (2025); Chen et al. (2025).

6.2 Conflict Compositions

A conflict occurs when one method destroys information needed by another, changes a shared representation or control signal, introduces an incompatible optimization objective, or adds execution overhead that removes the intended benefit Qin et al. (2024); Zhao and Zhao (2024); Chen et al. (2025). These conflicts are conditional. The same pair can be complementary at moderate settings and conflicting when compression becomes aggressive or when the implementation changes the interaction Guo et al. (2024); Qu et al. (2025).

Test whether recovery is possible before increasing recovery cost. Ultra-low-bit quantization can remove information that ordinary LoRA cannot reconstruct reliably Qin et al. (2024); Jeon et al. (2025). Measure the gap before and after recovery across precision, sparsity, or rank settings. If the unrecovered gap widens as recovery capacity increases, preserve more information in the first stage or co-train the compressed representation and recovery module instead of adding a larger adapter.

Separate optimization signals by their role. Joint optimization is not automatically superior because sparsity, quantization error, distillation quality, and architecture quality may favor different updates or model capacities Mishra and Marr (2017); Pham et al. (2023); Long et al. (2019); Liu et al. (2020); Kang et al. (2020). Quantization-aware KD can suffer from competing label and teacher objectives, while feature-KD gradients that improve retained weights can distort gradient-based pruning decisions Zhao and Zhao (2024); Yang et al. (2023). Ablate each loss separately and inspect both parameter updates and structural decisions. When a loss helps training but harms selection, use task gradients for pruning or architecture decisions and reserve the combined gradients for weight updates Yang et al. (2023).

Check representation and granularity compatibility. Independent pruning and quantization perturbations can accumulate, pruning can preserve outliers that are difficult to quantize, and independently trained LoRA updates may not fit the final quantization groups Frantar and Alistarh (2022); Guo et al. (2024); Xu et al. (2024). Compare both sequential orders with a shared reconstruction or aligned alternative. At aggressive sparsity or precision, use quantization-aware salience, compatible pruning and quantization groups, and adapter sharing that matches the deployed quantizer Guo et al. (2024); Qu et al. (2025); Xu et al. (2024).

Do not assume a larger joint search space is better. Heterogeneous subnetworks can interfere through shared weights, and compression can change the router or importance signals used by a conditional architecture Xu et al. (2022); Chen et al. (2025); Huang et al. (2025). Partition NAS candidates with substantially different size or convergence behavior. For MoE models, compare routing and sensitivity signals before and after expert compression, then recalibrate the router rather than using routing frequency as a universal pruning or precision score Chen et al. (2025); Huang et al. (2025).

Preserve trainability until trainable stages are complete. When low-bit conversion must interact with later adaptation, use a differentiable QAT formulation so the quantizer and recovery parameters can co-adapt, as demonstrated by L4Q Jeon et al. (2025). Use fixed PTQ as a terminal conversion when later stages do not need to update the compressed representation. If the adapter must be merged, test the merged

low-bit model rather than reporting only the unmerged training configuration Xu et al. (2024); Jeon et al. (2025).

Diagnose with matched ablations. Compare each method alone, both application orders, the sequential composition, and a joint or representation-aligned alternative at matched quality and resource levels Frantar and Alistarh (2022); Jeon et al. (2025). This comparison identifies whether the second method adds an independent benefit, repairs damage from the first stage, or creates avoidable interference. For sparse, low-rank, or routed models, include metadata, branch or expert loading, kernel launches, and end-to-end execution under the intended batch and sequence length Chen et al. (2025); Huang et al. (2025); Li et al. (2025).

6.3 Libraries for Composing Lightweight Methods

No single library covers the complete composition workflow. Table 8 classifies libraries into three groups depending on the major use scenarios: creating composition, executing an upstream composition, and serving an optimized LLM.

For creating the composition: Model Optimizer supports NVIDIA-oriented compression workflows, Intel Neural Compressor combines quantization, pruning, and KD, and torchao provides native PyTorch quantization and sparsity NVIDIA (2026a); Intel (2026a); PyTorch (2026b). DeepSpeed integrates distributed training with compression, while Olive searches and orders optimization passes across supported backends DeepSpeed (2026); Microsoft (2026a). NNCF prepares quantized or sparse models for OpenVINO, and Hugging Face Optimum routes models to hardware-specific optimization packages Intel (2026b); Hugging Face (2026a).

For executing an upstream composition: TensorRT compiles optimized models for NVIDIA hardware, ONNX Runtime provides execution through multiple hardware providers, and TVM supports cross-hardware graph and kernel compilation NVIDIA (2026b); Microsoft (2026b); Apache Software Foundation (2026). ExecuTorch and LiteRT execute upstream compositions on mobile and embedded devices PyTorch (2026a); Google (2026). Dynamic-shape behavior remains backend-specific because compiled profiles and mobile or NPU delegates can require bounded or static dimensions NVIDIA (2026b); Microsoft (2026b); PyTorch (2026a); Google (2026).

For serving an optimized LLM: TensorRT-LLM targets NVIDIA LLM inference, while vLLM and SGLang combine quantized execution with LoRA, cache management, batching, and distributed parallelism NVIDIA (2026c); vLLM Project (2026); SGLang Project (2026). llama.cpp provides portable single-node execution, and Ollama provides a local model-management layer around preconverted models ggml-org (2026); Ollama (2026). Triton schedules requests across existing inference backends, whereas the archived TGI project is retained primarily for existing deployments NVIDIA (2026d); Hugging Face (2026b).

7 Challenges and Future work

The composition space grows rapidly with the number of methods, application orders, coupling strategies, and hyperparameters. Because the best configuration also depends

Table 8: Compact support matrix for libraries that create, execute, or serve compositions of lightweight methods. Methods: Qt=quantization, Pr=pruning or sparsity, KD=knowledge distillation, and PEFT=parameter-efficient fine-tuning. ✓: direct support in the listed workflow role; ○: support through a companion package, backend, or upstream artifact; –: no documented method-specific support.

Library	Qt	Pr	KD	PEFT	Primary hardware
A. Create or search the composition					
NVIDIA Model Optimizer NVIDIA (2026a)	✓	✓	✓	✓	NVIDIA GPUs.
Intel Neural Compressor Intel (2026a)	✓	✓	✓	–	Intel CPUs, GPUs, and Gaudi accelerators.
PyTorch torchao PyTorch (2026b)	✓	✓	–	–	NVIDIA/AMD/Intel/Apple GPUs and ARM CPUs.
DeepSpeed DeepSpeed (2026)	✓	✓	✓	–	NVIDIA and AMD GPUs.
Microsoft Olive Microsoft (2026a)	✓	○	○	✓	Backend-dependent CPUs, GPUs, and NPUs.
OpenVINO + NNCF Intel (2026b)	✓	✓	–	–	Intel CPUs, GPUs, and NPUs.
Hugging Face Optimum Hugging Face (2026a)	○	○	○	○	Backend-dependent.
B. Compile or execute an upstream composition					
NVIDIA TensorRT NVIDIA (2026b)	✓	✓	○	○	NVIDIA GPUs and DLA accelerators.
ONNX Runtime Microsoft (2026b)	✓	○	–	–	CPUs, GPUs, and NPUs.
Apache TVM Apache Software Foundation (2026)	✓	○	○	–	CPUs, GPUs, DSPs, and custom accelerators.
ExecuTorch PyTorch (2026a)	✓	✓	–	–	Mobile CPUs, GPUs, NPUs, and MCUs.
Google LiteRT Google (2026)	✓	✓	○	○	Mobile and embedded CPUs, GPUs, and NPUs.
C. Serve or locally run an optimized LLM					
TensorRT-LLM NVIDIA (2026c)	✓	–	–	✓	NVIDIA GPUs.
Triton Inference Server NVIDIA (2026d)	○	○	○	○	CPUs, NVIDIA GPUs, and backend accelerators.
vLLM vLLM Project (2026)	✓	–	–	✓	CPUs, GPUs, TPUs, and supported accelerators.
SGLang SGLang Project (2026)	✓	–	–	✓	NVIDIA/AMD GPUs, Intel CPUs, TPUs, and Ascend.
llama.cpp ggml-org (2026)	✓	–	–	✓	CPUs and NVIDIA/AMD/Apple GPUs.
Ollama Ollama (2026)	○	–	–	✓	CPUs; NVIDIA, AMD, and Apple GPUs.
HF TGI Hugging Face (2026b)	✓	–	–	✓	NVIDIA GPUs, limited AMD/Gaudi support.

on the task, hardware, and runtime, exhaustive evaluation is infeasible. Future composition search should therefore jointly optimize method selection, ordering, and coupling against measured accuracy and deployment cost on the target platform.

Real-device studies report latency, throughput, and peak memory Ma et al. (2023); Li et al. (2025); Qin et al. (2024); Mozaffari et al. (2025), but measurements of DRAM traffic, cache behavior, memory bandwidth, and energy remain rare. Hardware counters or bus-level instrumentation can reveal when quantization, sparsity, and low-rank branches reduce data movement and when metadata or irregular access patterns offset their benefits.

LLM composition research still covers only a small part of the design space. Multimodal models, long-context inference, mixture-of-experts routing, and hardware-specific KV-cache management remain underexplored Park et al. (2024). Future studies should distinguish PEFT used only for training from adapters retained at deployment and report the cost of all adapters, sparse metadata, low-rank branches, and quantization parameters.

Automated pipelines for LLMs and multimodal models should minimize negative method interactions while optimizing latency, memory, energy, and adaptation cost. Reports should identify the explored methods, composition order, hardware and runtime, and measured accuracy and deployment cost. Controlled comparisons and theoretical analysis are also needed to explain why particular orders and coupling strategies succeed.

8 Conclusions

This survey treats the composition of lightweight methods as the unit of analysis and classifies compositions as sequential application, joint optimization, or hybrid composition according to their optimization dependencies. The reviewed evidence shows that no category is universally superior. Sequential application is appropriate when later stages can operate on fixed outputs, joint optimization is useful when compression variables or objectives interact, and hybrid composition supports joint adaptation followed by separate recovery or refinement. Across these categories, the benefit of composition depends on whether the methods address distinct bottlenecks, preserve trainability, and remain compatible at the target compression severity.

To further advance the field, more systematic research on combination strategies across different techniques and their application in diverse domains will be necessary. Such efforts will not only enhance model efficiency but also improve adaptability to real-world constraints, contributing to more effective and broadly applicable lightweight solutions.

Data availability

The results used in this survey are available in the studies cited in the article. The comparative values extracted or estimated from those studies are presented in the tables and accompanying text. No new experimental data were generated.

Acknowledgements

This research was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT)

(No. RS-2026-25617952, AI Star Fellowship Support) and by the research fund of Hanbat National University in 2026.

References

Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. ACM Journal on Emerging Technologies in Computing Systems (JETC) **13**(3), 1–18 (2017)

Augasta, M., Kathirvalavakumar, T.: Pruning algorithms of neural networks—a comparative study. Open Computer Science **3**(3), 105–115 (2013)

Apache Software Foundation: Apache TVM Documentation (2026). https://tvm.apache.org/docs/

Aghli, N., Ribeiro, E.: Combining weight pruning and knowledge distillation for cnn compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3191–3198 (2021)

Ba, J., Caruana, R.: Do deep nets really need to be deep? Advances in neural information processing systems **27** (2014)

Blalock, D., Gonzalez Ortiz, J.J., Frankle, J., Guttag, J.: What is the state of neural network pruning? Proceedings of machine learning and systems **2**, 129–146 (2020)

Blakeney, C., Huish, N., Yan, Y., Zong, Z.: Simon says: Evaluating and mitigating bias in pruned neural networks with knowledge distillation. arXiv preprint arXiv:2106.07849 (2021)

Bhalgat, Y., Lee, J., Nagel, M., Blankevoort, T., Kwak, N.: Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 696–697 (2020)

Baskin, C., Liss, N., Schwartz, E., Zheltonozhskii, E., Giryes, R., Bronstein, A.M., Mendelson, A.: Uniq: Uniform noise injection for non-uniform quantization of neural networks. ACM Transactions on Computer Systems (TOCS) **37**(1-4), 1–15 (2021)

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.*: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)

Bulat, A., Martinez, B., Tzimiropoulos, G.: Bats: Binary architecture search. In: European Conference on Computer Vision, pp. 309–325 (2020). Springer

Banner, R., Nahshan, Y., Soudry, D.: Post training 4-bit quantization of convolutional networks for rapid-deployment. Advances in Neural Information Processing Systems

32 (2019)

Beltagy, I., Peters, M.E., Cohan, A.: Longformer: The long-document transformer. arXiv:2004.05150 (2020)

Bercovich, A., Ronen, T., Abramovich, T., Ailon, N., Assaf, N., Dabbah, M., Galil, I., Geifman, A., Geifman, Y., Golan, I., *et al.*: Puzzle: Distillation-based NAS for inference-optimized LLMs. In: Proceedings of the 42nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 267, pp. 3806–3830 (2025)

Bai, H., Wu, J., King, I., Lyu, M.R.: Few shot network compression via cross distillation. CoRR **abs/1911.09450** (2019) 1911.09450

Choi, Y., Choi, J., El-Khamy, M., Lee, J.: Data-free network quantization with adversarial knowledge distillation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 710–711 (2020)

Chen, L., Chen, Y., Xi, J., Le, X.: Knowledge from the original network: restore a better pruned network with knowledge distillation. Complex & Intelligent Systems, 1–10 (2021)

Choi, Y., El-Khamy, M., Lee, J.: Learning sparse low-precision neural networks with learnable regularization. IEEE Access **8**, 96963–96974 (2020)

Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once for all: Train one network and specialize it for efficient deployment. In: International Conference on Learning Representations (2020)

Cai, Z., He, X., Sun, J., Vasconcelos, N.: Deep learning with low precision by half-wave gaussian quantization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5918–5926 (2017)

Choukroun, Y., Kravchik, E., Yang, F., Kisilev, P.: Low-bit quantization of neural networks for efficient inference. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pp. 3009–3018 (2019). IEEE

Cho, J., Lee, M.: Building a compact convolutional neural network for embedded intelligent sensor systems using group sparsity and knowledge distillation. Sensors **19**(19), 4307 (2019)

Chen, F., Li, S., Han, J., Ren, F., Yang, Z.: Review of lightweight deep convolutional neural networks. Archives of Computational Methods in Engineering **31**(4), 1915–1937 (2024)

Choudhary, T., Mishra, V., Goswami, A., Sarangapani, J.: A comprehensive survey on model compression and acceleration. Artificial Intelligence Review **53**, 5113–5155

(2020)

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. In: European Conference on Computer Vision (ECCV) (2020)

Chen, D., Mei, J.-P., Wang, C., Feng, Y., Chen, C.: Online knowledge distillation with diverse peers. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 3430–3437 (2020)

Chen, Y., Meng, G., Zhang, Q., Zhang, X., Song, L., Xiang, S., Pan, C.: Joint neural architecture search and quantization. arXiv preprint arXiv:1811.09426 (2018)

Chen, D., Mei, J.-P., Zhang, Y., Wang, C., Wang, Z., Feng, Y., Chen, C.: Cross-layer distillation with semantic calibration. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 7028–7036 (2021)

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., Sutskever, I.: Generative pretraining from pixels. In: International Conference on Machine Learning, pp. 1691–1703 (2020). PMLR

Chen, Y., Shao, Y., Wang, P., Cheng, J.: EAC-MoE: Expert-selection aware compressor for mixture-of-experts large language models. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 12942–12963. Association for Computational Linguistics, ??? (2025). https://doi.org/10.18653/v1/2025.acl-long.633

Cheng, J., Wang, P.-s., Li, G., Hu, Q.-h., Lu, H.-q.: Recent advances in efficient computation of deep convolutional neural networks. Frontiers of Information Technology & Electronic Engineering **19**, 64–77 (2018)

Cheng, Y., Wang, D., Zhou, P., Zhang, T.: A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282 (2017)

Cheng, Y., Wang, D., Zhou, P., Zhang, T.: Model compression and acceleration for deep neural networks: The principles, progress, and challenges. IEEE Signal Processing Magazine **35**(1), 126–136 (2018)

Chen, Y., Xie, Y., Yang, R., Jiang, W., Wang, W., He, Y., Chen, Y., Zhao, P., Wang, Y.: Collaborative compression for large-scale moe deployment on edge. arXiv preprint arXiv:2509.25689 (2025)

Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M.W., Keutzer, K.: Zeroq: A novel zero shot quantization framework. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 13169–13178 (2020)

Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task

and hardware. In: International Conference on Learning Representations (ICLR) (2019)

Cheng, H., Zhang, M., Shi, J.Q.: A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. IEEE Transactions on Pattern Analysis and Machine Intelligence (2024)

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: International Conference on Learning Representations (ICLR) (2021)

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Burstein, J., Doran, C., Solorio, T. (eds.) Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (2019). https://doi.org/10.18653/v1/N19-1423

DeepSpeed: Model Compression Documentation (2026). https://www.deepspeed.ai/tutorials/model-compression/

Deng, L., Li, G., Han, S., Shi, L., Xie, Y.: Model compression and hardware acceleration for neural networks: A comprehensive survey. Proceedings of the IEEE **108**(4), 485–532 (2020)

Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L.: QLoRA: Efficient finetuning of quantized LLMs. In: Advances in Neural Information Processing Systems, vol. 36 (2023)

Dong, X., Yang, Y.: Network pruning via transformable architecture search. Advances in Neural Information Processing Systems **32** (2019)

Esser, S.K., McKinstry, J.L., Bablani, D., Appuswamy, R., Modha, D.S.: Learned step size quantization. In: International Conference on Learning Representations (ICLR) (2020)

Frantar, E., Alistarh, D.: Optimal brain compression: A framework for accurate post-training quantization and pruning. In: Advances in Neural Information Processing Systems, vol. 35, pp. 4475–4488. Curran Associates, Inc., ??? (2022)

Faraone, J., Fraser, N., Blott, M., Leong, P.H.: Syq: Learning symmetric quantization for efficient deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4300–4309 (2018)

Frederick, T., Greg, M.: Deep neural network compression by in-parallel pruning-quantization. IEEE Transactions on Pattern Analysis and Machine Intelligence (2018)

Fang, J., Shafiee, A., Abdel-Aziz, H., Thorsley, D., Georgiadis, G., Hassoun, J.H.: Post-training piecewise linear quantization for deep neural networks. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16, pp. 69–86 (2020). Springer

Fan, A., Stock, P., Graham, B., Grave, E., Gribonval, R., Jegou, H., Joulin, A.: Training with quantization noise for extreme model compression. In: International Conference on Learning Representations (ICLR) (2021)

Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: International Conference on Machine Learning, pp. 1737–1746 (2015). PMLR

Gupta, M., Camci, E., Keneta, V.R., Vaidyanathan, A., Kanodia, R., James, A., Foo, C.-S., Wu, M., Lin, J.: Is complexity required for neural network pruning? a case study on global magnitude pruning. In: 2024 IEEE Conference on Artificial Intelligence (CAI), pp. 747–754 (2024). IEEE

ggml-org: llama.cpp Repository and Documentation (2026). https://github.com/ggml-org/llama.cpp

Guo, H., Greengard, P., Xing, E.P., Kim, Y.: LQ-LoRA: Low-rank plus quantized matrix decomposition for efficient language model finetuning. In: International Conference on Learning Representations (2024)

Gao, T., Guo, L., Zhao, S., Xu, P., Yang, Y., Liu, X., Wang, S., Zhu, S., Zhou, D.: QuantNAS: Quantization-aware neural architecture search for efficient deployment on mobile device. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 1704–1713 (2024)

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W., Keutzer, K.: A survey of quantization methods for efficient neural network inference. In: Low-Power Computer Vision, pp. 291–326. Chapman and Hall/CRC, ??? (2022)

Ghimire, D., Kil, D., Kim, S.-h.: A survey on efficient convolutional neural networks and hardware acceleration. Electronics **11**(6), 945 (2022)

Gong, R., Liu, X., Jiang, S., Li, T., Hu, P., Lin, J., Yu, F., Yan, J.: Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 4852–4861 (2019)

Guo, J., Li, Y., Lin, W., Chen, Y., Li, J.: Network decoupling: From regular to

depthwise separable convolutions. In: British Machine Vision Conference (2018)

Google: LiteRT Documentation (2026). https://ai.google.dev/edge/litert

Guo, J., Ouyang, W., Xu, D.: Multi-dimensional pruning: A unified framework for model compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1508–1517 (2020)

Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., Pang, R.: Conformer: Convolution-augmented transformer for speech recognition. In: Proc. Interspeech, pp. 5036–5040 (2020)

Guo, Y.: A survey on methods and theories of quantized neural networks. arXiv preprint arXiv:1808.04752 (2018)

Guo, J., Wu, J., Wang, Z., Liu, J., Yang, G., Ding, Y., Gong, R., Qin, H., Liu, X.: Compressing large language models by joint sparsification and quantization. In: Proceedings of the 41st International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 235, pp. 16945–16957 (2024)

Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient dnns. Advances in neural information processing systems **29** (2016)

Gou, J., Yu, B., Maybank, S.J., Tao, D.: Knowledge distillation: A survey. International Journal of Computer Vision **129**(6), 1789–1819 (2021)

Gao, S., Zhang, Y., Huang, F., Huang, H.: Bilevelpruning: Unified dynamic and static channel pruning for convolutional neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 16090–16100 (2024)

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16, pp. 544–560 (2020). Springer

Hawks, B., Duarte, J., Fraser, N.J., Pappalardo, A., Tran, N., Umuroglu, Y.: Ps and qs: Quantization-aware pruning for efficient low latency neural network inference. Frontiers in Artificial Intelligence **4**, 676564 (2021)

Hinton, G.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)

He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (2018). International Joint Conferences on Artificial Intelligence Organization

Han, D., Kim, J., Kim, J.: Deep pyramidal residual networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5927–5935 (2017)

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 784–800 (2018)

Huang, W., Liao, Y., Liu, J., He, R., Tan, H., Zhang, S., Li, H., Liu, S., Qi, X.: Mixture compressor for mixture-of-experts LLMs gains more. In: International Conference on Learning Representations (2025)

Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4700–4708 (2017)

Heo, B., Lee, M., Yun, S., Choi, J.Y.: Knowledge transfer via distillation of activation boundaries formed by hidden neurons. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 3779–3787 (2019)

Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In: International Conference on Learning Representations (ICLR) (2016)

Howard, A.G.: Mo-bilenets: Efficient convolutional neural networks for mo-bile vision applications. arXiv preprint arXiv:1704.04861 (2017)

Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. Advances in neural information processing systems **28** (2015)

Hu, P., Peng, X., Zhu, H., Aly, M.M.S., Lin, J.: Opq: Compressing deep neural networks with one-shot pruning-quantization. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 7780–7788 (2021)

Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7132–7141 (2018)

Hameed, M.G.A., Tahaei, M.S., Mosleh, A., Nia, V.P.: Convolutional neural network compression through generalized kronecker product decomposition. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, pp. 771–779 (2022)

Hugging Face: Optimum Documentation (2026). https://huggingface.co/docs/optimum/index

Hugging Face: Text Generation Inference Documentation. Repository archived in 2026

(2026). https://huggingface.co/docs/text-generation-inference/en/index

He, Y., Xiao, L.: Structured pruning for deep convolutional neural networks: A survey. IEEE transactions on pattern analysis and machine intelligence (2023)

Hua, W., Zhou, Y., De Sa, C.M., Zhang, Z., Suh, G.E.: Channel gating neural networks. Advances in Neural Information Processing Systems **32** (2019)

He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)

He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1389–1397 (2017)

Intel: Intel Neural Compressor Documentation (2026). https://intel.github.io/neural-compressor/latest/

Intel: OpenVINO and NNCF Model Optimization Documentation (2026). https://docs.openvino.ai/2026/openvino-workflow/model-optimization.html

Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37. ICML'15, pp. 448–456. JMLR.org, ??? (2015)

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2704–2713 (2018)

Jeon, H., Kim, Y., Kim, J.-J.: L4Q: Parameter efficient quantization-aware fine-tuning on large language models. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 2002–2024. Association for Computational Linguistics, ??? (2025). https://doi.org/10.18653/v1/2025.acl-long.99

Jin, Q., Ren, J., Zhuang, R., Hanumante, S., Li, Z., Chen, Z., Wang, Y., Yang, K., Tulyakov, S.: F8net: Fixed-point 8-bit only multiplication for network quantization. arXiv preprint arXiv:2202.05239 (2022)

Jung, S., Son, C., Lee, S., Son, J., Han, J.-J., Kwak, Y., Hwang, S.J., Choi, C.: Learning to quantize deep networks by optimizing quantization intervals with task loss. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4350–4359 (2019)

Jin, Q., Yang, L., Liao, Z.: Adabits: Neural network quantization with adaptive bit-widths. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2146–2156 (2020)

Kim, J., Bhalgat, Y., Lee, J., Patel, C., Kwak, N.: Qkd: Quantization-aware knowledge distillation. arXiv preprint arXiv:1911.12491 (2019)

Kim, J., Chang, S., Kwak, N.: Pqk: model compression via pruning, quantization, and knowledge distillation. arXiv preprint arXiv:2106.14681 (2021)

Kim, Y.J., Fahim, R., Awadalla, H.H.: Mixture of quantized experts (moqe): Complementary effect of low-bit quantization and robustness. arXiv preprint arXiv:2310.02410 (2023)

Klein, A., Golebiowski, J., Ma, X., Perrone, V., Archambeau, C.: Structural pruning of large language models via neural architecture search. In: AutoML Conference 2023 (2023)

Kang, M., Han, B.: Operation-aware soft channel pruning using differentiable masks. In: International Conference on Machine Learning, pp. 5122–5131 (2020). PMLR

Kulkarni, U., Hosamani, A.S., Masur, A.S., Hegde, S., Vernekar, G.R., Chandana, K.S.: A survey on quantization methods for optimization of deep neural networks. In: 2022 International Conference on Automation, Computing and Renewable Systems (ICACRS), pp. 827–834 (2022). IEEE

Kitaev, N., Kaiser, L., Levskaya, A.: Reformer: The efficient transformer. In: International Conference on Learning Representations (ICLR) (2020)

Kim, D., Lee, J., Ham, B.: Distance-aware quantization. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 5271–5280 (2021)

Kulkarni, U., Meena, S., Gurlahosur, S.V., Bhogar, G.: Quantization friendly mobilenet (qf-mobilenet) architecture for vision based applications on embedded platforms. Neural Networks **136**, 28–39 (2021)

Kang, M., Mun, J., Han, B.: Towards oracle knowledge distillation with neural architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 4404–4411 (2020)

Kim, J., Park, S., Kwak, N.: Paraphrasing complex network: Network compression via factor transfer. Advances in neural information processing systems **31** (2018)

Kim, Y., Rush, A.M.: Sequence-level knowledge distillation. arXiv preprint arXiv:1606.07947 (2016)

Kaleem, S.M., Rouf, T., Habib, G., Lall, B., et al.: A comprehensive review of knowledge distillation in computer vision. arXiv preprint arXiv:2404.00936 (2024)

Long, X., Ben, Z., Zeng, X., Liu, Y., Zhang, M., Zhou, D.: Learning sparse convolutional neural network via quantization with low rank regularization. IEEE Access **7**, 51866–51876 (2019)

Liu, Y., Cao, J., Li, B., Hu, W., Maybank, S.: Learning to explore distillability and sparsability: A joint framework for model compression. IEEE Transactions on Pattern Analysis and Machine Intelligence **45**(3), 3378–3395 (2023) https://doi.org/10.1109/TPAMI.2022.3185317

Leng, C., Dou, Z., Li, H., Zhu, S., Jin, R.: Extremely low bit neural network: Squeeze the last bit out with admm. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)

Li, Y., Gu, S., Mayer, C., Gool, L.V., Timofte, R.: Group sparsity: The hinge between filter pruning and decomposition for network compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8018–8027 (2020)

Li, Y., Gong, R., Tan, X., Yang, Y., Hu, P., Zhang, Q., Yu, F., Wang, W., Gu, S.: Brecq: Pushing the limit of post-training quantization by block reconstruction. In: International Conference on Learning Representations (ICLR) (2021)

Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X.: Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing **461**, 370–403 (2021)

Lin, S., Ji, R., Chen, C., Tao, D., Luo, J.: Holistic cnn compression via low-rank decomposition with knowledge transfer. IEEE transactions on pattern analysis and machine intelligence **41**(12), 2889–2905 (2018)

Liu, Y., Jia, X., Tan, M., Vemulapalli, R., Zhu, Y., Green, B., Wang, X.: Search to distill: Pearls are everywhere but not the eyes. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7539–7548 (2020)

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., Yan, X.: Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. Advances in neural information processing systems **32** (2019)

Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: International Conference on Learning Representations (2017)

Lee, S.H., Kim, D.H., Song, B.C.: Self-supervised knowledge distillation using singular value decomposition. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 335–350 (2018)

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 10012–10022 (2021)

Li, Y., Lin, S., Liu, J., Ye, Q., Wang, M., Chao, F., Yang, F., Ma, J., Tian, Q., Ji, R.: Towards compact cnns via collaborative compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 6438–6447 (2021)

Li, T., Li, J., Liu, Z., Zhang, C.: Few sample knowledge distillation for efficient network compression. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 14627–14635 (2020). https://doi.org/10.1109/CVPR42600.2020.01465

Li, Z., Li, H., Meng, L.: Model compression for deep neural networks: A survey. Computers **12**(3), 60 (2023)

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2736–2744 (2017)

Li, Z., Liu, F., Yang, W., Peng, S., Zhou, J.: A survey of convolutional neural networks: analysis, applications, and prospects. IEEE transactions on neural networks and learning systems **33**(12), 6999–7019 (2021)

Liu, Y., Lin, Z., Yuan, F.: Rosita: Refined bert compression with integrated techniques. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 8715–8722 (2021)

Li, M., Lin, Y., Zhang, Z., Cai, T., Li, X., Guo, J., Xie, E., Meng, C., Zhu, J.-Y., Han, S.: SVDQuant: Absorbing outliers by low-rank components for 4-bit diffusion models. In: International Conference on Learning Representations (2025)

Liebenwein, L., Maalouf, A., Feldman, D., Rus, D.: Compressing neural networks: Towards determining the optimal layer-wise decomposition. Advances in Neural Information Processing Systems **34**, 5328–5344 (2021)

Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.-T., Sun, J.: Metapruning: Meta learning for automatic neural network channel pruning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 3296–3305 (2019)

Liu, Z., Oguz, B., Zhao, C., Chang, E., Stock, P., Mehdad, Y., Shi, Y., Krishnamoorthi, R., Chandra, V.: LLM-QAT: Data-free quantization aware training for large language models. In: Findings of the Association for Computational Linguistics: ACL 2024 (2024)

Liu, X., Parhi, K.K.: Tensor decomposition for model reduction in neural networks: A review. IEEE Circuits and Systems Magazine **23**(2), 8–28 (2023)

Li, C., Peng, J., Yuan, L., Wang, G., Liang, X., Lin, L., Chang, X.: Block-wisely supervised neural architecture search with knowledge distillation. In: Proceedings

of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1989–1998 (2020)

Liu, Y., Shu, Z., Li, Y., Lin, Z., Perazzi, F., Kung, S.-Y.: Content-aware gan compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12156–12166 (2021)

Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. In: International Conference on Learning Representations (ICLR) (2019)

Liu, J., Tripathi, S., Kurup, U., Shah, M.: Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey. arXiv preprint arXiv:2005.04275 (2020)

Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. In: International Conference on Learning Representations (ICLR) (2018)

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F.E.: A survey of deep neural network architectures and their applications. Neurocomputing **234**, 11–26 (2017)

Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., Cheng, K.-T.: Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 722–737 (2018)

Li, B., Wu, B., Su, J., Wang, G.: Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In: European Conference on Computer Vision, pp. 639–654 (2020). Springer

Li, Y., Yu, Y., Liang, C., Karampatziakis, N., He, P., Chen, W., Zhao, T.: LoftQ: LoRA-fine-tuning-aware quantization for large language models. In: International Conference on Learning Representations (2024)

Li, Z., Yuan, G., Niu, W., Zhao, P., Li, Y., Cai, Y., Shen, X., Zhan, Z., Kong, Z., Jin, Q., *et al.*: Npas: A compiler-aware framework of unified network pruning and architecture search for beyond real-time mobile acceleration. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 14255–14266 (2021)

Li, Y., Yuan, G., Wen, Y., Hu, J., Evangelidis, G., Tulyakov, S., Wang, Y., Ren, J.: Efficientformer: Vision transformers at mobilenet speed. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems, vol. 35, pp. 12934–12949. Curran Associates, Inc., ??? (2022)

Li, Y., Yu, Y., Zhang, Q., Liang, C., He, P., Chen, W., Zhao, T.: Losparse: Structured compression of large language models based on low-rank and sparse approximation.

In: International Conference on Machine Learning, pp. 20336–20350 (2023). PMLR

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 19–34 (2018)

Mi, J.-X., Feng, J., Huang, K.-Y.: Designing efficient convolutional neural network structure: A survey. Neurocomputing **489**, 139–156 (2022)

Ma, X., Fang, G., Wang, X.: LLM-Pruner: On the structural pruning of large language models. In: Advances in Neural Information Processing Systems, vol. 36, pp. 21702–21720 (2023)

Mishra, R., Gupta, H.P., Dutta, T.: A survey on deep neural network compression: Challenges, overview, and solutions. arXiv preprint arXiv:2010.03954 (2020)

Microsoft: Olive Model Optimization Documentation (2026). https://microsoft.github.io/Olive/

Microsoft: ONNX Runtime Model Optimization Documentation (2026). https://onnxruntime.ai/docs/performance/model-optimizations/

Movva, R., Lei, J., Longpre, S., Gupta, A., DuBois, C.: Combining compressions for multiplicative size scaling on natural language tasks. arXiv preprint arXiv:2208.09684 (2022)

Mishra, A., Marr, D.: Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. arXiv preprint arXiv:1711.05852 (2017)

Muralidharan, S., Sreenivas, S.T., Joshi, R., Chochowski, M., Patwary, M., Shoeybi, M., Catanzaro, B., Kautz, J., Molchanov, P.: Compact language models via pruning and knowledge distillation. In: Advances in Neural Information Processing Systems (2024)

Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. In: International Conference on Learning Representations (ICLR) (2017)

Macko, V., Weill, C., Mazzawi, H., Gonzalvo, J.: Improving neural architecture search image classifiers via ensemble learning. arXiv preprint arXiv:1903.06236 (2019)

Mao, Y., Wang, Y., Wu, C., Zhang, C., Wang, Y., Yang, Y., Zhang, Q., Tong, Y., Bai, J.: Ladabert: Lightweight adaptation of bert through hybrid model compression. arXiv preprint arXiv:2004.04124 (2020)

Mozaffari, M., Yazdanbakhsh, A., Dehnavi, M.M.: SLiM: One-shot quantization and sparsity with low-rank approximation for LLM weight compression. In: Proceedings of the 42nd International Conference on Machine Learning (2025)

Ma, N., Zhang, X., Zheng, H.-T., Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 116–131 (2018)

Nagel, M., Amjad, R.A., Van Baalen, M., Louizos, C., Blankevoort, T.: Up or down? adaptive rounding for post-training quantization. In: International Conference on Machine Learning, pp. 7197–7206 (2020). PMLR

Nagel, M., Baalen, M.v., Blankevoort, T., Welling, M.: Data-free quantization through weight equalization and bias correction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 1325–1334 (2019)

Neill, J.O.: An overview of neural network compression. arXiv preprint arXiv:2006.03669 (2020)

Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., Van Baalen, M., Blankevoort, T.: A white paper on neural network quantization. arXiv preprint arXiv:2106.08295 (2021)

Nan, K., Liu, S., Du, J., Liu, H.: Deep model compression for mobile platforms: A survey. Tsinghua Science and Technology **24**(6), 677–693 (2019)

NVIDIA: NVIDIA Model Optimizer Documentation (2026). https://nvidia.github.io/Model-Optimizer/

NVIDIA: TensorRT Documentation (2026). https://docs.nvidia.com/deeplearning/tensorrt/latest/index.html

NVIDIA: TensorRT-LLM Documentation (2026). https://github.com/NVIDIA/TensorRT-LLM

NVIDIA: Triton Inference Server Documentation (2026). https://github.com/triton-inference-server/server

Ollama: Ollama Documentation (2026). https://docs.ollama.com/

Oguntola, I., Olubeko, S., Sweeney, C.: Slimnets: An exploration of deep model compression and acceleration. arXiv preprint arXiv:1808.00496 (2018)

Park, S., Choi, J., Lee, S., Kang, U.: A Comprehensive Survey of Compression Algorithms for Language Models (2024)

Peng, H., Du, H., Yu, H., Li, Q., Liao, J., Fu, J.: Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. Advances in Neural Information Processing Systems **33** (2020)

Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via

parameters sharing. In: International Conference on Machine Learning, pp. 4095–4104 (2018). PMLR

Pham, C., Hoang, T., Do, T.-T.: Collaborative multi-teacher knowledge distillation for learning low bit-width deep neural networks. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 6435–6443 (2023)

Prakosa, S.W., Leu, J.-S., Chen, Z.-H.: Improving the accuracy of pruned network using knowledge distillation. Pattern Analysis and Applications **24**(2), 819–830 (2021)

Phan, H., Liu, Z., Huynh, D., Savvides, M., Cheng, K.-T., Shen, Z.: Binarizing mobilenet via evolution-based searching. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 13420–13429 (2020)

Park, J., No, A.: Prune your model before distill it. In: European Conference on Computer Vision, pp. 120–136 (2022). Springer

Polino, A., Pascanu, R., Alistarh, D.: Model compression via distillation and quantization. In: International Conference on Learning Representations (2018)

Peng, H., Wu, J., Chen, S., Huang, J.: Collaborative channel pruning for deep networks. In: International Conference on Machine Learning, pp. 5113–5122 (2019). PMLR

Passban, P., Wu, Y., Rezagholizadeh, M., Liu, Q.: Alp-kd: Attention-based layer projection for knowledge distillation. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 13657–13665 (2021)

PyTorch: ExecuTorch Documentation (2026). https://docs.pytorch.org/executorch/stable/

PyTorch: torchao Documentation (2026). https://docs.pytorch.org/ao/stable/

Qu, X., Aponte, D., Banbury, C., Robinson, D.P., Ding, T., Koishida, K., Zharkov, I., Chen, T.: Automatic joint structured pruning and quantization for efficient neural network training and compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 15234–15244 (2025)

Qin, D., Leichner, C., Delakis, M., Fornoni, M., Luo, S., Yang, F., Wang, W., Banbury, C., Ye, C., Akin, B., Aggarwal, V., Zhu, T., Moro, D., Howard, A.: MobileNetV4: Universal models for the mobile ecosystem. In: European Conference on Computer Vision (2024)

Qin, H., Ma, X., Zheng, X., Li, X., Zhang, Y., Liu, S., Luo, J., Liu, X., Magno, M.: Accurate LoRA-finetuning quantization of LLMs via information retention. In: Proceedings of the 41st International Conference on Machine Learning. Proceedings

of Machine Learning Research, vol. 235, pp. 41498–41516. PMLR, ??? (2024)

Rokh, B., Azarpeyvand, A., Khanteymoori, A.: A comprehensive survey on model quantization for deep neural networks in image classification. ACM Transactions on Intelligent Systems and Technology **14**(6), 1–50 (2023)

Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y.: Fitnets: Hints for thin deep nets. In: International Conference on Learning Representations (ICLR) (2015)

Radford, A., Narasimhan, K.: Improving language understanding by generative pre-training. (2018)

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., Sutskever, I.: Zero-shot text-to-image generation. In: International Conference on Machine Learning, pp. 8821–8831 (2021). Pmlr

Ren, A., Zhang, T., Ye, S., Li, J., Xu, W., Qian, X., Lin, X., Wang, Y.: Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 925–938 (2019)

Saragadam, V., Balestriero, R., Veeraraghavan, A., Baraniuk, R.G.: Deeptensor: Low-rank tensor decomposition with deep network priors. IEEE Transactions on Pattern Analysis and Machine Intelligence (2024)

Song, G., Chai, W.: Collaborative learning for deep neural networks. Advances in neural information processing systems **31** (2018)

Scardapane, S., Comminiello, D., Hussain, A., Uncini, A.: Group sparse regularization for deep neural networks. Neurocomputing **241**, 81–89 (2017)

Sze, V., Chen, Y.-H., Yang, T.-J., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE **105**(12), 2295–2329 (2017)

Sheng, T., Feng, C., Zhuo, S., Zhang, X., Shen, L., Aleksic, M.: A quantization-friendly separable convolution for mobilenets. In: 2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2), pp. 14–18 (2018). IEEE

SGLang Project: SGLang Documentation (2026). https://docs.sglang.io/

Shen, M., Han, K., Xu, C., Wang, Y.: Searching for accurate binary neural architectures. In: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, pp. 0–0 (2019)

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510–4520 (2018)

Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31 (2017)

Sainath, T.N., Kingsbury, B., Sindhwani, V., Arisoy, E., Ramabhadran, B.: Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6655–6659 (2013). IEEE

Shim, K., Lee, M., Choi, I., Boo, Y., Sung, W.: Svd-softmax: Fast softmax approximation on large vocabulary neural networks. Advances in neural information processing systems **30** (2017)

Shen, M., Liang, F., Gong, R., Li, Y., Li, C., Lin, C., Yu, F., Yan, J., Ouyang, W.: Once quantization-aware training: High performance extremely low-bit architecture search. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 5340–5349 (2021)

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)

Saha, R., Sagan, N., Srivastava, V., Goldsmith, A.J., Pilanci, M.: Compressing large language models using low rank and low precision decomposition. In: Advances in Neural Information Processing Systems (2024)

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016)

Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2820–2828 (2019)

Tay, Y., Dehghani, M., Bahri, D., Metzler, D.: Efficient transformers: A survey. arXiv preprint arXiv:2009.06732 (2020)

Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: Proceedings of the 36th International Conference on Machine Learning

(ICML), pp. 6105–6114 (2019). PMLR

Tu, C.-H., Lee, J.-H., Chan, Y.-M., Chen, C.-S.: Pruning depthwise separable convolutions for mobilenet compression. In: 2020 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2020). https://doi.org/10.1109/IJCNN48605.2020.9207259

Tung, F., Mori, G.: Clip-q: Deep network compression learning by in-parallel pruning-quantization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7873–7882 (2018)

Tung, F., Mori, G.: Similarity-preserving knowledge distillation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 1365–1374 (2019)

Tai, C., Xiao, T., Zhang, Y., Wang, X., *et al.*: Convolutional neural networks with low-rank regularization. In: International Conference on Learning Representations (ICLR) (2015)

Vadera, S., Ameen, S.: Methods for pruning deep neural networks. IEEE Access **10**, 63280–63300 (2022)

Vaswani, A.: Attention is all you need. Advances in Neural Information Processing Systems (2017)

vLLM Project: vLLM Documentation (2026). https://docs.vllm.ai/en/stable/

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 10734–10742 (2019)

Weng, O.: Neural network quantization for efficient inference: A survey. arXiv preprint arXiv:2112.06126 (2021)

Wang, H., Gui, S., Yang, H., Liu, J., Wang, Z.: Gan slimming: All-in-one gan compression by a unified optimization framework. In: European Conference on Computer Vision, pp. 54–73 (2020). Springer

Wang, Z., Lin, J., Geng, X., Aly, M.M.S., Chandrasekhar, V.: Rdo-q: Extremely fine-grained channel-wise quantization via rate-distortion optimization. In: European Conference on Computer Vision, pp. 157–172 (2022). Springer

Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: Haq: Hardware-aware automated quantization with mixed precision. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8612–8620 (2019)

Wu, J., Leng, C., Wang, Y., Hu, Q., Cheng, J.: Quantized convolutional neural networks for mobile devices. In: Proceedings of the IEEE Conference on Computer

Vision and Pattern Recognition, pp. 4820–4828 (2016)

Wu, Q., Lu, X., Xue, S., Wang, C., Wu, X., Fan, J.: Sbnn: Slimming binarized neural network. Neurocomputing **401**, 113–122 (2020)

Wu, M., Ma, W., Li, Y., Zhao, X.: The optimization method of knowledge distillation based on model pruning. In: 2020 Chinese Automation Congress (CAC), pp. 1386–1390 (2020). IEEE

Wei, L., Ma, Z., Yang, C., Yao, Q.: Advances in the neural network quantization: A comprehensive review. Applied Sciences **14**(17), 7445 (2024)

Wei, Y., Pan, X., Qin, H., Ouyang, W., Yan, J.: Quantization mimic: Towards very tiny cnn for object detection. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 267–283 (2018)

Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y., Han, S.: Apq: Joint search for network architecture, pruning and quantization policy. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2078–2087 (2020)

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., Zhou, M.: Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. Advances in Neural Information Processing Systems **33**, 5776–5788 (2020)

Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. Advances in neural information processing systems **29** (2016)

Wang, L., Yoon, K.-J.: Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. IEEE transactions on pattern analysis and machine intelligence **44**(6), 3048–3068 (2021)

Xu, S., Huang, A., Chen, L., Zhang, B.: Convolutional neural network pruning: A survey. In: 2020 39th Chinese Control Conference (CCC), pp. 7458–7463 (2020). IEEE

Xie, H., Jiang, W., Luo, H., Yu, H.: Model compression via pruning and knowledge distillation for person re-identification. Journal of Ambient Intelligence and Humanized Computing **12**(2), 2149–2161 (2021)

Xu, D., Mukherjee, S., Liu, X., Dey, D., Wang, W., Zhang, X., Awadallah, A., Gao, J.: Few-shot task-agnostic neural architecture search for distilling large language models. In: Advances in Neural Information Processing Systems, vol. 35, pp. 28644–28656. Curran Associates, Inc., ??? (2022)

Xu, Y., Xie, L., Gu, X., Chen, X., Chang, H., Zhang, H., Chen, Z., Zhang, X., Tian, Q.: QA-LoRA: Quantization-aware low-rank adaptation of large language models.

In: International Conference on Learning Representations (2024)

Xie, Y., Zhang, Z., Zhou, D., Xie, C., Song, Z., Liu, X., Wang, Y., Lin, X., Xu, A.: MoE-Pruner: Pruning Mixture-of-Experts Large Language Model using the Hints from Its Router (2024)

Yao, Z., Aminabadi, R.Y., Zhang, M., Wu, X., Li, C., He, Y.: Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In: Advances in Neural Information Processing Systems, vol. 35 (2022)

Yuan, Y., Chen, C., Hu, X., Peng, S.: Towards low-bit quantization of deep neural networks with limited data. In: 2020 25th International Conference on Pattern Recognition (ICPR), pp. 4377–4384 (2021). IEEE

Yang, Z., Cui, Y., Yao, X., Wang, S.: Gradient-based intra-attention pruning on pre-trained language models. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 2775–2790. Association for Computational Linguistics, ??? (2023). https://doi.org/10.18653/v1/2023.acl-long.156

Yang, H., Gui, S., Zhu, Y., Liu, J.: Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2178–2188 (2020)

Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H.: Netadapt: Platform-aware neural network adaptation for mobile applications. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 285–300 (2018)

Yim, J., Joo, D., Bae, J., Kim, J.: A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4133–4141 (2017)

Yu, R., Li, A., Chen, C.-F., Lai, J.-H., Morariu, V.I., Han, X., Gao, M., Lin, C.-Y., Davis, L.S.: Nisp: Pruning networks using neuron importance score propagation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9194–9203 (2018)

Ye, J., Lu, X., Lin, Z., Wang, J.Z.: Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In: International Conference on Learning Representations (2019)

Yu, X., Liu, T., Wang, X., Tao, D.: On compressing deep models by low rank and sparse decomposition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7370–7379 (2017)

Yao, L., Pi, R., Xu, H., Zhang, W., Li, Z., Zhang, T.: Joint-detnas: Upgrade your detector with nas, pruning and dynamic distillation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10175–10184 (2021)

Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., Hua, X.-s.: Quantization networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7308–7316 (2019)

Yao, W., Zhang, J., Li, C., Li, S., He, L., Zhang, B.: Semantic segmentation optimization algorithm based on knowledge distillation and model pruning. In: 2019 2nd International Conference on Artificial Intelligence and Big Data (ICAIBD), pp. 261–265 (2019). IEEE

Ye, S., Zhang, T., Zhang, K., Li, J., Xie, J., Liang, Y., Liu, S., Lin, X., Wang, Y.: A unified framework of dnn weight pruning and weight clustering/quantization using admm. arXiv preprint arXiv:1811.01907 (2018)

Zhang, L., Bao, C., Ma, K.: Self-distillation: Towards efficient and compact neural networks. IEEE Transactions on Pattern Analysis and Machine Intelligence **44**(8), 4388–4403 (2021)

Zhang, X., Colbert, I., Das, S.: Learning low-precision structured subnetworks using joint layerwise channel pruning and uniform quantization. Applied Sciences **12**(15), 7829 (2022)

Zhang, M., Chen, H., Shen, C., Yang, Z., Ou, L., Yu, X., Zhuang, B.: LoRAPrune: Structured pruning meets low-rank parameter-efficient fine-tuning. In: Findings of the Association for Computational Linguistics: ACL 2024 (2024)

Zhu, X., Gong, S., et al.: Knowledge distillation by on-the-fly native ensemble. Advances in neural information processing systems **31** (2018)

Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontañón, S., Pham, P., Ravula, A., Wang, Q., Yang, L., Ahmed, A.: Big bird: Transformers for longer sequences. CoRR **abs/2007.14062** (2020) 2007.14062

Zhao, R., Hu, Y., Dotzel, J., De Sa, C., Zhang, Z.: Improving neural network quantization without retraining using outlier channel splitting. In: International Conference on Machine Learning, pp. 7543–7552 (2019). PMLR

Zagoruyko, S., Komodakis, N.: Wide residual networks. In: Proceedings of the British Machine Vision Conference (BMVC) (2016)

Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: International Conference on Learning Representations (ICLR) (2017)

Zhang, H., Liu, L., Huang, Y., Yang, Z., Lei, X., Wen, B.: CaKDP: Category-aware

knowledge distillation and pruning framework for lightweight 3d object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 15331–15341 (2024)

Zhao, C., Ni, B., Zhang, J., Zhao, Q., Zhang, W., Tian, Q.: Variational convolutional neural network pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2780–2789 (2019)

Zhang, C., Peng, Y.: Better and faster: Knowledge transfer from multiple self-supervised learning tasks via graph distillation for video classification. In: Lang, J. (ed.) Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, pp. 1135–1141. ijcai.org, ??? (2018). https://doi.org/10.24963/IJCAI.2018/158

Zhang, S., Papyan, V.: OATS: Outlier-aware pruning through sparse and low rank decomposition. In: International Conference on Learning Representations (2025)

Zhang, L., Song, J., Gao, A., Chen, J., Bao, C., Ma, K.: Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 3713–3722 (2019)

Zhuang, B., Shen, C., Tan, M., Liu, L., Reid, I.: Towards effective low-bitwidth convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7920–7928 (2018)

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)

Zhang, Y., Xiang, T., Hospedales, T.M., Lu, H.: Deep mutual learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4320–4328 (2018)

Zhang, D., Yang, J., Ye, D., Hua, G.: Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 365–382 (2018)

Zhao, K., Zhao, M.: Self-supervised quantization-aware knowledge distillation. In: Proceedings of the 27th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 238, pp. 4375–4383. PMLR, ??? (2024)

Zhang, X., Zou, J., He, K., Sun, J.: Accelerating very deep convolutional networks for classification and detection. IEEE transactions on pattern analysis and machine intelligence **38**(10), 1943–1955 (2015)

Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6848–6856 (2018)

Zhang, X., Zou, J., Ming, X., He, K., Sun, J.: Efficient and accurate approximations of nonlinear convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1984–1992 (2015)