Prune-Quantize-Distill: An Ordered Pipeline for Efficient Neural Network Compression

1st Longsheng Zhou
Institute of Advanced Technology
University of Science and Technology of China
Hefei, China
zlongsheng@mail.ustc.edu.cn

2nd Yu Shen
Supercomputing Center
University of Science and Technology of China
Hefei, China
shenyu@ustc.edu.cn

***Abstract*—Modern deployment often requires trading accuracy for efficiency under tight CPU and memory constraints, yet common compression proxies such as parameter count or FLOPs do not reliably predict wall-clock inference time. In particular, unstructured sparsity can reduce model storage while failing to accelerate (and sometimes slightly slowing down) standard CPU execution due to irregular memory access and sparse-kernel overhead. Motivated by this gap between *compression* and *acceleration*, we study a practical, ordered pipeline that targets *measured* latency by combining three widely used techniques: unstructured pruning, INT8 quantization-aware training (QAT), and knowledge distillation (KD). Empirically, INT8 QAT provides the dominant runtime benefit, while pruning mainly acts as a capacity-reduction pre-conditioner that improves the robustness of subsequent low-precision optimization; KD, applied last, recovers accuracy within the already constrained sparse INT8 regime without changing the deployment form. We evaluate on CIFAR-10/100 using three backbones (ResNet-18, WRN-28-10, and VGG-16-BN). Across all settings, the ordered pipeline achieves a stronger accuracy–size–latency frontier than any single technique alone, reaching 0.99–1.42 ms CPU latency with competitive accuracy and compact checkpoints. Controlled ordering ablations with a fixed 20/40/40 epoch allocation further confirm that stage order is consequential, with the proposed ordering generally performing best among the tested permutations. Overall, our results provide a simple guideline for edge deployment: evaluate compression choices in the joint accuracy–size–latency space using measured runtime, rather than proxy metrics alone.**

***Index Terms*—Quantization, Network pruning, Knowledge Distillation, Inference Speed, Multi-step Compression Process**

I. Introduction

Deep neural networks (DNNs) have achieved strong performance in vision, language, and multimodal tasks, but modern models are often over-parameterized and expensive to run. This becomes a real obstacle on resource-constrained platforms such as mobile devices, embedded systems, and edge accelerators, where memory footprint, power budget, and inference latency are all limited. In these settings, simply shrinking the backbone (fewer layers or channels) can quickly lead to noticeable accuracy degradation.

Model compression provides a more controlled way to trade a small amount of accuracy for substantial efficiency gains [1]. Recent surveys summarize a wide range of compression techniques and emphasize that deployment constraints (hardware backend, operator support, and runtime overhead) often dominate practical outcomes [2]. A comparative study by Marinò *et al.* further suggests that different compression families typically improve different parts of the accuracy–efficiency space and that hybrid strategies are often preferable to single techniques [3].

In this work, we focus on three widely used building blocks: pruning, quantization, and knowledge distillation (KD). Pruning removes redundant parameters and can substantially reduce effective capacity with limited accuracy loss under appropriate retraining [4], while its practical speed benefits depend on sparsity structure and hardware support [5]. Quantization reduces precision and storage cost, enabling efficient inference on standard integer backends (e.g., INT8) [6]. KD transfers knowledge from a high-capacity teacher to a constrained student and is often used to recover accuracy after compression [7], [8]. We deliberately restrict ourselves to these standard components, aiming for a minimal and reproducible pipeline rather than introducing specialized operators or elaborate training tricks.

Despite steady progress, combining these tools in a reliable way is still tricky. Many hybrid approaches are tailored to particular architectures or datasets, or rely on tightly coupled joint objectives and careful hyperparameter tuning [9]. Other hybrid ideas target different model families with different deployment bottlenecks, which may not directly carry over to standard CNN inference [10]. Meanwhile, practitioners often prefer simple recipes built from standard components, because they are easier to reproduce and integrate into existing toolchains [11]. This motivates a concrete question: can we design a minimal hybrid pipeline that consistently improves the joint accuracy–size–latency trade-off on mainstream convolutional backbones under measured CPU runtime, without specialized sparse kernels or intricate training tricks?

We propose a fixed three-stage recipe: *global unstructured pruning → INT8 quantization-aware training (QAT) → knowledge distillation*, where the ordering is part of the method and targets a consistent deployable form (sparse INT8). Unstructured pruning alone often yields limited wall-clock benefits on general-purpose CPUs *without specialized sparse kernels*, but it reduces the active weight set and stabilizes subsequent low-precision optimization. INT8 QAT contributes most of the

*Corresponding author: shenyu@ustc.edu.cn

latency reduction on standard backends, and KD is applied last to recover accuracy after the model has entered the constrained sparse INT8 regime. Overall, the stages play complementary roles, and we validate the importance of ordering via a controlled stage permutation study.

We evaluate the pipeline on three backbone–dataset pairs: ResNet-18 on CIFAR-10, WRN-28-10 on CIFAR-100, and VGG-16-BN on CIFAR-10. We compare against pruning-only, quantization-only, and KD-only baselines, and analyze the joint accuracy–size–latency space using both scalar metrics and multi-dimensional visualizations. In addition, to align with common co-compression benchmarks in prior work, we report a literature-aligned comparison on ResNet-20/CIFAR-10 using relative BOPs.

Our main contributions are summarized as follows:

- **A minimal ordered co-compression recipe with role separation.** We propose a simple three-stage pipeline—global unstructured pruning → INT8 QAT → KD—built from standard components and evaluated at a consistent deployable endpoint (sparse INT8), avoiding reliance on specialized sparse kernels.
- **Controlled evidence that stage ordering is consequential.** Keeping the same ingredients, the same training budget, and the same deployable form, we perform a controlled stage-order ablation by permuting the three stages, showing that ordering alone can lead to clear accuracy differences while leaving latency within a narrow range.
- **Deployment-driven evaluation with consistent gains across settings.** We evaluate compression choices in the joint accuracy–size–latency space using measured CPU runtime, demonstrating that proxy metrics (e.g., parameter/FLOP reductions) may not reflect wall-clock behavior on standard backends. Across three CNN backbones and two datasets, the proposed ordering yields consistently favorable trade-offs over single-stage baselines; we additionally report a literature-aligned comparison using relative BOPs on ResNet-20/CIFAR-10 to corroborate the conclusions under a common proxy metric.

II. Related Work

We review the three standard ingredients used in our pipeline—pruning, quantization, and knowledge distillation—and then briefly discuss hybrid, multi-stage recipes.

A. Pruning

Pruning removes parameters (or structures) that contribute little to the final prediction. Early magnitude-based pruning with retraining showed that CNNs can often be made much smaller with only minor accuracy loss. Later work studied pruning together with quantization, including structured or coordinated pruning–quantization schemes [12]. In our setting, we care more about preserving accuracy on compact backbones than about relying on sparse kernels for speed. We therefore use global unstructured pruning mainly as an accuracy-friendly way to reduce the number of active weights before quantization, rather than treating pruning as the main source of wall-clock acceleration [13].Recent analyses further suggest that unstructured sparsity mainly reduces storage and may not translate to latency gains on general-purpose CPUs unless supported by specialized kernels, which motivates using pruning primarily as a capacity regularizer in our pipeline.

B. Quantization

Quantization speeds up inference by using low-precision integers for weights and activations. Early studies showed that quantized approximations can reduce test-time cost while keeping accuracy reasonably close to full precision [14]. Incremental schemes then pushed pretrained CNNs to low precision more gently, reducing the drop in accuracy. Deep Compression combined trained quantization with coding to further shrink the stored model [15]. A common issue is that aggressive post-training discretization can be brittle. For this reason, we use INT8 quantization-aware training (QAT) and optimize the model directly under fake-quant constraints, starting from a pruned initialization.Empirically, a pruned initialization can also ease low-precision optimization by lowering the effective noise accumulation, making INT8 training more stable than quantizing a dense model directly.

C. Knowledge Distillation

Knowledge distillation (KD) trains a student to match a stronger teacher, commonly through a KL loss on softened outputs together with cross-entropy. KD is often used to boost small or compressed models, including pruned networks, and can recover part of the accuracy lost during compression [16]. KD itself does not change size or latency, but it is a practical tool for "repairing" a student after pruning/quantization. We therefore apply KD at the end, inside the sparse INT8 space, to regain accuracy without changing the deployment cost.This "post-compression refinement" view is especially useful when quantization introduces functional shifts that are hard to correct by fine-tuning alone.

D. Hybrid and Multi-Stage Compression Frameworks

Many recent works combine multiple compression steps rather than relying on a single technique. Beyond the choice of ingredients (pruning/quantization/distillation), existing methods also differ substantially in their deployment assumptions and optimization paradigms, e.g., fixed-bit versus mixed-precision quantization, structured versus unstructured pruning, and staged pipelines versus constrained (white-box) formulations or coupled update rules [17]. To make these design choices explicit, we summarize representative paradigms in Table I. In contrast to tightly coupled co-optimization approaches that often rely on joint objectives and careful tuning, we adopt a minimal and strictly ordered recipe: global unstructured pruning → INT8 quantization-aware training (QAT) → knowledge distillation in the compressed space. This design favors fixed-bit INT8 deployment and uses KD as a lightweight accuracy recovery step without changing deployment cost. Overall, our framework aligns with the broader trend of staged

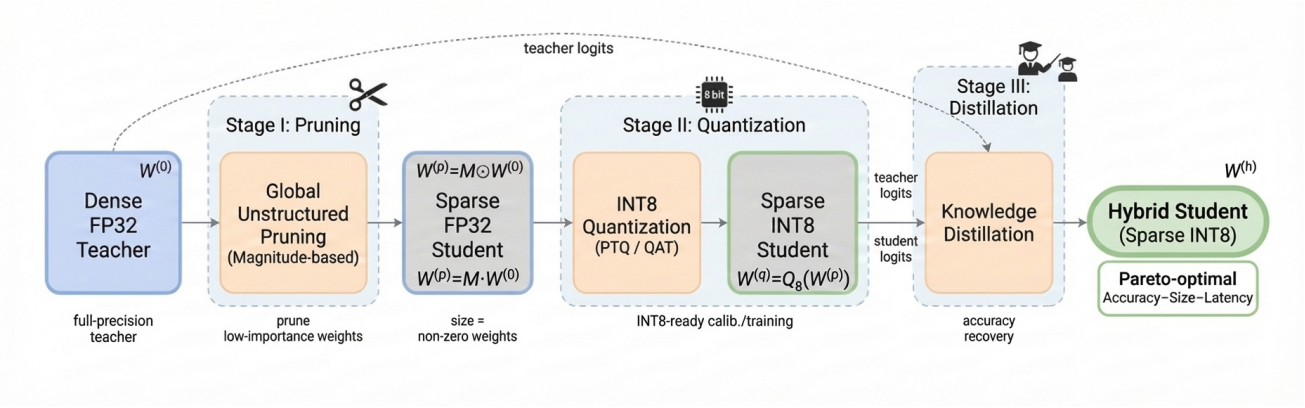


Fig. 1: Overview of the proposed hybrid compression pipeline. A dense FP32 teacher with weights $\mathbf{W}^{(0)}$ is first pruned into a sparse FP32 student ($\mathbf{W}^{(p)} = \mathbf{M} \odot \mathbf{W}^{(0)}$), then optimized under INT8 quantization (PTQ/QAT) to obtain a sparse INT8 student $\mathbf{W}^{(q)} = Q_8(\mathbf{W}^{(p)})$, and finally refined via knowledge distillation to produce the hybrid student $\mathbf{W}^{(h)}$ in the compressed space.

TABLE I: Design philosophy comparison (compact).

Method	INT8	Prune	Opt.	KD
Ours (P50→INT8→KD)	✓	Unstr.	Pipeline	✓
GETA	△	**Str.**	Joint	×
SQL	×	Unstr.	Constr.	×
QST	×	Unstr.	Coupled	×
ANNC	△	Unstr.	Constr.	×

Notes: INT8: ✓ fixed single-precision INT8; × primarily mixed-precision / bit-allocation; △ configuration-dependent. Str./Unstr.: structured/unstructured pruning. Opt.: Constr.=constrained (white-box) optimization.

compression, but emphasizes a clearer separation of the roles of pruning, quantization, and distillation, thereby characterizing how different combinations and ordering choices affect the final accuracy–efficiency trade-off.

III. Method

A. Problem Formulation and Hybrid View

We use three metrics throughout the paper: accuracy, effective model size, and inference latency. Effective size is measured by the number of non-zero parameters; we also report the serialized checkpoint footprint (MB) as a practical storage proxy (Tables II–III). Fig. 1 summarizes the proposed three-stage procedure.

Let $f(\mathbf{x}; \mathbf{W})$ denote a classifier trained on $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$. Standard supervised learning minimizes

$$\min_{\mathbf{W}} \ \mathcal{L}(\mathbf{W}) = \frac{1}{N} \sum_{i=1}^{N} \ell\big(f(\mathbf{x}_i; \mathbf{W}), y_i\big), \tag{1}$$

where $\ell(\cdot)$ is the cross-entropy loss.

Deployment imposes sparsity, INT8-deployability, and latency constraints:

$$\min_{\mathbf{W}} \ \mathcal{L}_{\text{task}}(\mathbf{W}) \quad \text{s.t.} \quad \mathbf{W} \in \mathcal{S}_\rho, \ \mathbf{W} \in \mathcal{Q}_8, \ \text{Lat}(\mathbf{W}) \le \tau, \tag{2}$$

where $\mathcal{S}_\rho$ is the set of models with target sparsity ρ, $\mathcal{Q}_8$ denotes INT8-deployable models, and τ is a latency budget.

Directly solving the above constrained optimization problem is inconvenient because sparsity is non-smooth and INT8 quantization is discrete. A useful way to summarize the interaction between the three stages is the coupled objective

$$\mathbf{W}^\star \approx \arg\min_{\mathbf{W}} \mathcal{L}_{\text{task}}\big(Q_8(\mathbf{M} \odot \mathbf{W})\big) + \lambda_s \|\mathbf{M} \odot \mathbf{W}\|_0 + \lambda_d \, \mathcal{L}_{KD}, \tag{3}$$

where $\mathbf{M} \in \{0,1\}^{|\mathbf{W}|}$ is a pruning mask and $\mathcal{L}_{KD}$ is defined in Sec. III-D.

In practice, we approximate this coupled objective with a fixed three-stage pipeline:

$$\mathbf{W}^{(0)} \xrightarrow{\text{Pruning}} \mathbf{W}^{(p)} \xrightarrow{\text{QAT}} \mathbf{W}^{(q)} \xrightarrow{\text{KD}} \mathbf{W}^{(h)}, \tag{4}$$

which can be interpreted as moving the model into progressively smaller feasible sets:

$$\mathbb{R}^{|\mathbf{W}|} \supset \mathcal{S}_\rho \supset \mathcal{S}_\rho \cap \mathcal{Q}_8. \tag{5}$$

The ordering matters because the final stage operates inside the deployable sparse INT8 space, rather than correcting errors in an unconstrained FP32 model.

B. Stage I: Global Unstructured Magnitude Pruning

We employ global unstructured magnitude pruning. With a binary mask $\mathbf{M}$,

$$\mathbf{W}^{(p)} = \mathbf{M} \odot \mathbf{W}^{(0)}, \qquad \|\mathbf{W}^{(p)}\|_0 = \sum_j M_j, \tag{6}$$

and the mask keeps the top $(1-\rho)$ weights by magnitude:

$$M_j = \mathbb{I}\big(|W_j^{(0)}| \ge \gamma_\rho\big), \quad \frac{\|\mathbf{W}^{(p)}\|_0}{\|\mathbf{W}^{(0)}\|_0} = 1 - \rho, \tag{7}$$

where γ_ρ is the threshold for sparsity ρ. A common first-order argument relates pruning W_j to the loss change:

$$\Delta\mathcal{L} \approx \left| \frac{\partial \mathcal{L}}{\partial W_j} W_j \right|. \tag{8}$$

Why unstructured pruning? Structured pruning can give direct speedups, but it often degrades accuracy on compact backbones. Unstructured pruning typically preserves accuracy better, even though it may not reduce CPU latency due to

irregular memory access. In this paper it is used mainly to reduce the active weight set and to make the later INT8 optimization less noisy.

C. Stage II: INT8 Quantization-Aware Training (QAT)

On top of $\mathbf{W}^{(p)}$, we perform INT8 QAT using uniform affine quantization:

$$Q_8(x) = \text{clip}\Big(\Big\lfloor \frac{x}{s} \Big\rceil + z,\ 0,\ 255\Big), \qquad \hat{x} = s\,(Q_8(x) - z), \tag{9}$$

where s is the scale and z is the zero-point. QAT minimizes the task loss under fake-quant constraints,

$$\min_{\mathbf{W}}\ \mathcal{L}_{\text{task}}\Big(f\big(\mathbf{x}; Q_8(\mathbf{M} \odot \mathbf{W})\big)\Big), \tag{10}$$

and uses the straight-through estimator (STE),

$$\frac{\partial Q_8(x)}{\partial x} \approx 1. \tag{11}$$

a) Quantization on top of pruning: Pruning reduces the number of active weights,

$$\|\mathbf{W}^{(p)}\|_0 = (1-\rho)\|\mathbf{W}^{(0)}\|_0, \tag{12}$$

and QAT quantizes the remaining weights to INT8,

$$\mathbf{W}^{(q)} = Q_8(\mathbf{W}^{(p)}). \tag{13}$$

Ignoring sparse storage overhead, this yields an approximate multiplicative size reduction:

$$\frac{\text{Size}(\mathbf{W}^{(q)})}{\text{Size}(\mathbf{W}^{(0)})} \approx (1-\rho)\cdot\frac{8}{32}, \qquad \text{Compr.} \approx \frac{32}{8}\cdot\frac{1}{1-\rho}. \tag{14}$$

b) Why pruning can stabilize QAT: Pruning may also reduce the aggregate effect of quantization perturbations by shrinking the active set. Under a standard uniform error model $Q_8(w) = w + \epsilon$ with $\epsilon \sim \mathcal{U}(-\Delta/2, \Delta/2)$,

$$\mathbb{E}\|\boldsymbol{\epsilon}\|_2^2 \le \frac{\Delta^2}{12}\cdot\|\mathbf{W}^{(p)}\|_0. \tag{15}$$

This provides an intuitive motivation (not a guarantee): fewer active weights can reduce the accumulated perturbation during INT8 optimization, which aligns with our empirical ordering ablation.

c) PTQ vs. QAT: We mainly report QAT since it is typically more robust than PTQ under aggressive compression. PTQ remains a training-free deployment option on the same compressed backbone.

D. Stage III: Knowledge Distillation for Accuracy Recovery

Pruning and quantization introduce a functional deviation from the dense teacher:

$$\Delta f(\mathbf{x}) = f(\mathbf{x}; \mathbf{W}^{(q)}) - f(\mathbf{x}; \mathbf{W}^{(0)}). \tag{16}$$

KD uses the dense teacher to guide the fake-quantized student. With logits $\mathbf{z}_t$ and $\mathbf{z}_s$,

$$\mathcal{L}_{KD} = \alpha\,\mathcal{L}_{CE} + (1-\alpha)T^2\,\text{KL}(\sigma(\mathbf{z}_t/T)\,\|\,\sigma(\mathbf{z}_s/T))\,, \tag{17}$$

where T is temperature and α balances CE and KD.

KD refines the student within the sparse INT8 feasible set:

$$\mathbf{W}^{(h)} = \arg\min_{\mathbf{W}\in\mathcal{S}_\rho\cap\mathcal{Q}_8} \mathbb{E}_{\mathbf{x}}\|f(\mathbf{x};\mathbf{W}) - f(\mathbf{x};\mathbf{W}^{(0)})\|_2^2, \tag{18}$$

and does not change effective size or latency:

$$\|\mathbf{W}^{(h)}\|_0 = \|\mathbf{W}^{(q)}\|_0, \quad \text{Lat}(\mathbf{W}^{(h)}) \approx \text{Lat}(\mathbf{W}^{(q)}). \tag{19}$$

In practice, we distill from the original dense FP32 teacher $\mathbf{W}^{(0)}$ to avoid propagating quantization artifacts from a low-precision teacher. We found logit-based distillation to be sufficient for compact CNN backbones, since it directly targets the decision boundary shift induced by pruning and INT8 constraints (Eq. 16). KD is applied with the same fake-quant modules enabled as in QAT, so the student is optimized in the exact deployable space rather than in an unconstrained surrogate. This differs from distilling before quantization, where improvements may partially vanish after discretization. Unless otherwise stated, we keep the KD setup lightweight (teacher fixed, student initialized from $\mathbf{W}^{(q)}$) and tune only (α, T). We also note that applying KD earlier in the pipeline, such as before pruning or before quantization, leads to less consistent gains in our experiments. When KD is performed on a dense or FP32 model, the distilled knowledge may not be preserved after subsequent pruning or discretization. By contrast, performing KD as the final stage allows the student to adapt its predictions to the combined effects of sparsity and INT8 quantization. From an optimization perspective, KD acts as a local refinement step that reshapes the loss landscape within the constrained feasible region. This makes it particularly effective for recovering accuracy without reintroducing additional parameters or increasing inference cost.

E. Complementarity and Ordering

The stages play complementary roles: pruning shrinks the active set and stabilizes subsequent INT8 optimization, QAT provides the main speedup, and KD recovers accuracy after the model has entered the sparse INT8 regime. Importantly, the *order* of these stages matters: under a controlled ordering ablation where we fix the same components and stage budgets (20/40/40 epochs) and only permute the stage order, the default ordering (Prune→QAT→KD) consistently achieves the best accuracy across backbones (Table II). Overall, the resulting sparse INT8 model occupies a strong region in the joint accuracy–size–latency space.

IV. Experiments

A. Experimental Setup

We conduct our main experiments on three backbone–dataset pairs: ResNet-18 on CIFAR-10, WideResNet-28-10 (WRN-28-10) on CIFAR-100, and VGG-16-BN on CIFAR-10. In addition, to align with common co-compression benchmarks in prior work, we report a literature-aligned comparison on ResNet-20/CIFAR-10. For each backbone, we first train a dense FP32 network as the *teacher* and as the reference point for reporting compression and speedup. Unless otherwise stated, the FP32 baseline is optimized for 100 epochs using SGD with cosine learning-rate decay.

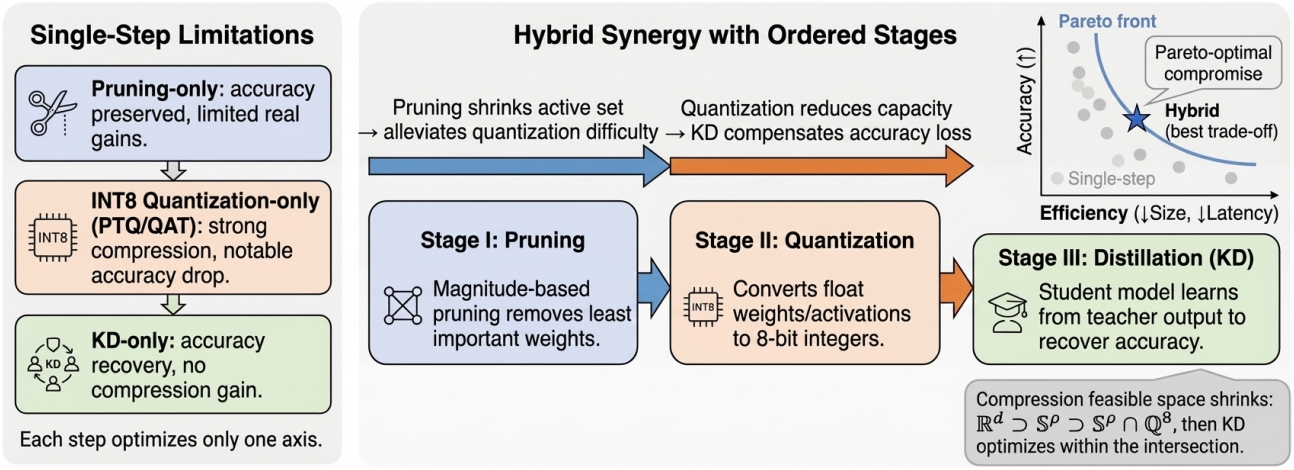


Fig. 2: Synergy of the ordered hybrid method. Left: limitations of single-step pruning-only, quantization-only, and KD-only strategies, each optimizing mainly one axis (accuracy or efficiency). Right: our three-stage ordering, where pruning shrinks the active set, quantization converts sparsity into real efficiency gains, and KD recovers accuracy in the compressed space, moving the model toward the Pareto front in the accuracy–efficiency plane.

a) Budgets and reporting protocols: We report results under three complementary protocols. **(i) Fully trained (Table II):** all methods are compared under a fixed total budget of 100 epochs. For the multi-stage pipeline, we allocate epochs across stages while keeping the total budget fixed (20/40/40 for Prune/QAT/KD). **(ii) Diagnostic snapshot (Table III):** an early-training snapshot used to diagnose the roles and interactions of individual stages in a multi-stage pipeline, rather than to claim final performance. **FT Epochs** denotes the number of fine-tuning epochs performed *on top of the baseline checkpoint*; for multi-stage pipelines, fine-tuning epochs are counted and summed across stages. **(iii) Literature-aligned benchmark (Table IV):** ResNet-20 on CIFAR-10, reported with accuracy and relative BOPs to match common co-compression settings.

Reading guide: cross-backbone conclusions (including ordering ablations) are drawn from the fully trained protocol (Table II);single-backbone diagnostic discussions use the snapshot protocol (Table III); and the literature-aligned benchmark (Table IV) is used only for comparison under standard co-compression reporting.

Starting from the same FP32 teacher initialization, we derive four variants: (i) pruning-only, (ii) QAT-only, (iii) KD-only, and (iv) the ordered hybrid pipeline (Prune→QAT→KD). We further include an **ordering ablation** by permuting the three stages while fixing the same components and the same 20/40/40 stage budgets (Table II).

b) Latency measurement: Inference latency is measured end-to-end on a single Intel Xeon (Skylake) CPU server under a unified PyTorch setup. All INT8 models are evaluated with the `fbgemm` backend, and FP32 models run on the same CPU under identical threading settings. We fix PyTorch to use 10 CPU threads, switch models to evaluation mode, and time forward passes with `time.perf_counter()` inside `torch.inference_mode()`. Each latency number is averaged over 100 repeated runs after 10 warm-up runs; we reuse the same test batch across repeats to reduce batch-content variance.

Effective model size is measured by the number of non-zero parameters and the serialized checkpoint footprint (in MB), which we denote as *Size (MB)* in the tables.

B. Main Results: Accuracy–Size–Latency Trade-off

Table II reports our main results under the fully trained protocol (fixed 100-epoch budget) on three backbone–dataset pairs. Across all settings, the ordered hybrid pipeline consistently achieves a favorable accuracy–size–latency trade-off compared with any single-stage baseline (as shown in Fig. 2). In particular, pruning alone tends to reduce the serialized footprint but yields limited CPU speedup, whereas INT8 QAT provides the most reliable latency reduction; combining pruning, INT8 QAT, and KD in the proposed order improves the final accuracy at essentially the same sparse-INT8 deployment form.

All methods are evaluated at a consistent deployable endpoint (sparse INT8) and under a fixed training budget, so the observed differences mainly reflect optimization outcomes rather than implementation choices. Overall, the gains are stable across both residual (ResNet/WRN) and VGG-style architectures, supporting the generality of the proposed recipe. **Controlled ordering ablation.** To isolate the effect of stage sequencing, we keep the same ingredients (pruning, INT8 QAT, and KD) and the same per-stage budgets (20/40/40), and only permute the stage order (Table II). The default ordering (Prune→QAT→KD) consistently yields the best accuracy across backbones, while moving pruning to the end (QAT→KD→Prune) causes the largest degradation. Notably, latency remains within a narrow range under the same sparse-INT8 deployment setting, so the accuracy gaps directly reflect

TABLE II: Main results (fully trained) across backbones and datasets with controlled stage-order ablation.

Method	ResNet-18 + CIFAR-10				WRN-28-10 + CIFAR-100				VGG-16-BN + CIFAR-10			
	Acc. (%) ↑	Size (MB) ↓	Lat. (ms) ↓	Speedup ↑	Acc. (%) ↑	Size (MB) ↓	Lat. (ms) ↓	Speedup ↑	Acc. (%) ↑	Size (MB) ↓	Lat. (ms) ↓	Speedup ↑
(a) Main results												
Baseline	78.37	42.65	2.45	1.00	76.03	139.38	3.42	1.00	79.38	56.18	2.62	1.00
Prune (30%)	79.84	33.13	2.43	1.01	**79.41**	97.56	3.43	1.00	80.11	39.33	2.64	0.99
Prune (50%)	**80.38**	26.97	2.55	0.96	79.07	69.69	3.36	1.02	80.55	28.11	2.81	0.93
QAT (INT8)	77.42	10.66	**0.99**	**2.47**	71.29	35.81	**1.42**	**2.41**	77.23	14.05	1.01	2.59
KD	76.33	42.65	2.49	0.98	72.06	139.38	3.34	1.02	78.57	56.18	2.69	0.97
Hybrid (Prune50% → QAT → KD)	79.62	**6.74**	1.00	2.45	78.69	**17.41**	1.42	2.41	**81.42**	**7.03**	**0.99**	**2.65**
(b) Stage-order ablation (same components, same 20/40/40 budgets)												
Hybrid (Prune50% → QAT → KD) *(default)*	**79.62**	**6.74**	**1.00**	**2.45**	**78.69**	**17.41**	1.42	2.41	**81.42**	**7.03**	0.99	2.65
Hybrid (Prune50% → KD → QAT)	79.00	6.74	1.02	2.40	78.20	17.41	**1.41**	**2.43**	80.90	7.03	**0.98**	**2.67**
Hybrid (QAT → Prune50% → KD)	78.50	6.74	1.00	2.45	77.40	17.41	1.43	2.39	80.30	7.03	0.99	2.65
Hybrid (QAT → KD → Prune50%)	76.60	6.74	1.01	2.43	75.10	17.41	1.42	2.41	78.80	7.03	1.00	2.62

Legend: **Bold**/underline indicate the best/second-best entries within each backbone block, computed separately for (a) and (b); ties are broken in favor of the *default* ordering in (b).

Notes: All hybrid variants use the same stage budgets (20/40/40 for Prune/QAT/KD). Speedup is relative to the FP32 baseline; minor differences may arise from rounding.

the impact of ordering. We next provide a diagnostic analysis to further explain why this ordering improves optimization in the sparse INT8 regime.

C. Why Ordering Matters: Diagnostic Analysis on ResNet-18/CIFAR-10

We use the diagnostic snapshot on ResNet-18/CIFAR-10 (Table III) to analyze stage interactions and ordering effects, rather than to claim final performance. Fig. 3 visualizes the resulting accuracy–efficiency trade-offs.

A key hardware observation is that **unstructured pruning does not necessarily translate to wall-clock speedups on standard CPUs**. For example, 50% pruning reduces the serialized footprint (42.65→26.97 MB) but slightly increases latency (2.45→2.55 ms), indicating that pruning mainly acts as capacity reduction and a pre-conditioner for subsequent optimization. In contrast, **INT8 QAT provides the dominant latency reduction** (0.99 ms, 2.48× speedup), yet it is harder to optimize in the early snapshot and suffers the largest accuracy drop (Val. 73.98→69.63). These observations motivate separating the roles of stages: pruning for capacity/conditioning, QAT for latency, and KD for accuracy recovery.

The ordered pipeline resolves this tension. Compared with QAT-only, the *Hybrid* pipeline preserves essentially the same latency (1.00 vs. 0.99 ms) while recovering accuracy to 76.91% (+7.28 points), yielding a substantially stronger accuracy–size–latency trade-off. Moreover, it achieves the highest compression ratio among all compared variants (6.33×), highlighting that the gains come with a compact deployable model. The 3D view in Fig. 4c further illustrates how the proposed ordering shifts the solution toward a more favorable region in the joint trade-off space. As auxiliary diagnostics, ROC/PR curves in Fig. 4 show consistent trends with the main accuracy–efficiency comparisons.

D. Literature-aligned Comparison on ResNet-20/CIFAR-10

Following the standard training protocol used in prior co-compression studies, we report a literature-aligned comparison on ResNet-20/CIFAR-10 in Table IV, using accuracy and relative BOPs as commonly adopted in this line of work.

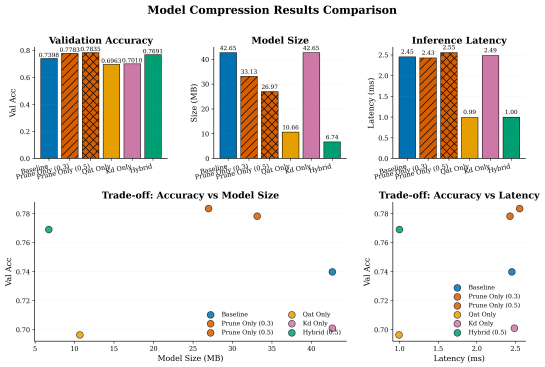


Fig. 3: Comparison of standalone baselines and the ordered hybrid pipeline on ResNet-18/CIFAR-10.

TABLE III: Diagnostic snapshot on ResNet-18/CIFAR-10.

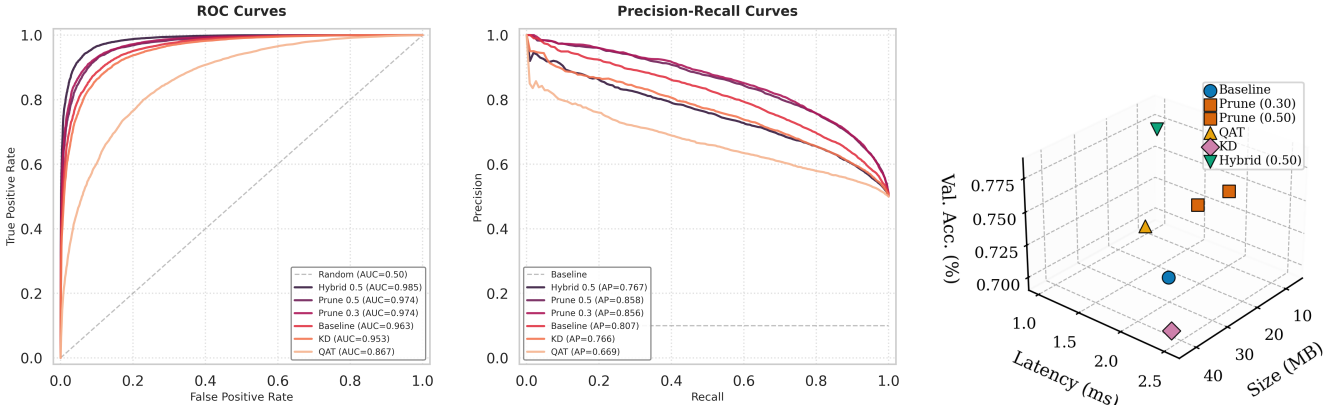
Method	Ratio	FT	Acc. (%) ↑		Efficiency			
			Tr.	Val.	Size (MB) ↓	Compr. (×) ↑	Lat. (ms) ↓	Spdup (×) ↑
Baseline	–	20	75.51	73.98	42.65	1.00	2.45	1.00
Prune Only	30%	10	81.75	77.83	33.13	1.29	2.43	1.01
Prune Only	50%	10	**81.77**	**78.35**	26.97	1.58	2.55	0.96
QAT Only	–	8	72.42	69.63	10.66	4.00	**0.99**	**2.48**
KD Only	–	10	70.41	70.10	42.65	1.00	2.49	0.99
Hybrid	50%	28	80.67	76.91	**6.74**	**6.33**⋆	1.00	2.46

Notes: **FT** = fine-tuning epochs; *Size* = checkpoint MB. ⋆ best compression.

As shown in Table IV, our ordered pipeline attains strong accuracy (91.83%) while achieving the lowest relative BOPs (3.1) among the compared methods. Notably, this trade-off is obtained with a simple and deployable recipe (50% unstructured pruning with W8/A8 INT8 QAT and final-stage KD), rather than a heavily coupled mixed-precision objective. Together with the deployment-driven CPU latency results in Table II, these findings suggest that the proposed ordering remains competitive under both measured runtime and literature-aligned proxy metrics.

V. Conclusion

We study an ordered compression pipeline—Prune→INT8 QAT→KD—and show that its stages play complementary roles. On standard CPUs, unstructured pruning alone does



(a) ROC curves. (b) Precision–recall curves. (c) 3D accuracy–size–latency trade-off.

Fig. 4: Global analysis on ResNet-18/CIFAR-10: ROC/PR curves and the 3D accuracy–size–latency trade-off.

TABLE IV: Literature-aligned comparison on ResNet-20/CIFAR-10.

Method	**W-bit**	**Prune**	**Acc. (%) ↑**	**Rel. BOPs ↓**
Baseline	32	None	91.70	100.0
SQL	MP	Unstr.	90.90	6.1
QST-P	MP	Unstr.	91.80	5.0
GETA	MP	Str.	91.42	4.5
QST-Q	MP	Unstr.	91.60	3.3
Ours (P50→INT8QAT→KD)	**8**	Unstr.	**91.83**	**3.1**

Notes: MP denotes mixed-precision. Rel. BOPs follow each paper's convention; ours uses W8/A8 INT8 QAT with 50% unstructured sparsity.

not guarantee wall-clock speedups, but it reduces capacity and stabilizes subsequent low-precision optimization by shrinking the active weight set. INT8 QAT contributes most of the latency reduction, while KD, applied last, recovers accuracy within the already constrained sparse INT8 regime. Under a fixed training budget and a consistent deployable endpoint (sparse INT8), we observe consistently favorable accuracy–size–latency trade-offs across ResNet-18, WRN-28-10, and VGG-16-BN on CIFAR-10/100 compared with single-stage baselines, and controlled ordering ablations further confirm that stage sequencing is consequential. Overall, our results highlight a practical takeaway: deployment decisions should be guided by measured latency alongside accuracy, rather than parameter/FLOP counts alone. We further show that the proposed ordering remains competitive under a literature-aligned proxy metric (relative BOPs) on ResNet-20/CIFAR-10. Finally, the pipeline is modular and can incorporate alternative pruning criteria, quantization schemes, or distillation losses without changing the overall recipe.

Future work will explore hardware-friendly structured sparsity and more automated policy selection to better map Pareto frontiers under different deployment constraints.

REFERENCES

[1] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, 2018.

[2] P. V. Dantas, W. Sabino da Silva, L. C. Cordeiro, and C. B. Carvalho, "A comprehensive review of model compression techniques in machine learning," *Appl. Intell.*, vol. 54, no. 22, p. 11804–11844, Sep. 2024. [Online]. Available: https://doi.org/10.1007/s10489-024-05747-w

[3] G. C. Marinó, A. Petrini, D. Malchiodi, and M. Frasca, "Deep neural networks compression: A comparative survey and choice recommendations," *Neurocomputing*, vol. 520, pp. 152–170, 2023.

[4] T. Frederick and M. Greg, "Deep neural network compression by in-parallel pruning-quantization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 6, 2018.

[5] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 12, pp. 10 558–10 578, 2024.

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

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

[8] Y. Qian, X. Li, J. Cao, J. Zhang, H. Li, and J. Chen, "Boosting pruned networks with linear over-parameterization," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*. IEEE, 2024, pp. 5070–5074.

[9] J. Kim, S. Chang, and N. Kwak, "Pqk: model compression via pruning, quantization, and knowledge distillation," *arXiv preprint arXiv:2106.14681*, 2021.

[10] L. Yu, W. Xiang, K. Han, G. Liu, and R. Kompella, "Comp-diff: A unified pruning and distillation framework for compressing diffusion models," *IEEE Trans. Multimedia*, vol. 27, pp. 8486–8497, 2025.

[11] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[12] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[13] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2018.

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

[15] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," *arXiv preprint arXiv:1902.08153*, 2019.

[16] Y. Tian, D. Krishnan, and P. Isola, "Contrastive representation distillation," *arXiv preprint arXiv:1910.10699*, 2019.

[17] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.