Journal of Computer Science and Electrical Engineering
Print ISSN: 2663-1938
Online ISSN: 2663-1946
DOI: https://doi.org/10.61784/jcsee4154

ACCURACY-AWARE MODEL COMPRESSION FOR REAL-TIME VISUAL PERCEPTION UNDER RESOURCE CONSTRAINTS

FengYuan Zhang[1*], JieSi Yang[2]
[1]*New York University, New York, NY 10003, USA.*
[2]*National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan Region, China.*
[*]*Corresponding Author: FengYuan Zhang*

Abstract: Deploying real-time visual perception models on CPU-only edge devices requires compression that respects measured latency while preserving task accuracy. Uniform quantization and pruning ignore the uneven sensitivity of network components, and existing allocation methods optimize a single compression dimension with additive per-layer cost models that miss the precision-switch overheads of CPU inference runtimes. This paper presents ACORN, a post-training, accuracy-aware compression framework that jointly allocates channel-pruning ratios and numerical precision to the components of a pretrained detector under explicit latency and memory budgets. ACORN estimates the accuracy cost of every option as a perturbation of the uniformly quantized deployment model with a confidence-weighted discrepancy restricted to task-relevant categories, predicts latency from per-node profiles of the runtime-optimized graph augmented with pairwise quantize/dequantize costs, prunes channels without training through ridge least-squares reconstruction, and solves the allocation exactly as an integer linear program. On YOLOv8n deployed for PASCAL VOC detection on a single CPU core with ONNX Runtime, ACORN reaches 63.47% mAP@[.5:.95] at the latency of uniform INT8 quantization (62.77%), within 0.07 points of the FP32 model while running 2.28× faster with a 3.6× smaller model. At a 2.51× speedup it retains 63.17%, whereas latency-aware knapsack pruning and uniform DepGraph pruning at similar latency fall to 52.54% and 30.33%. The complete allocation uses 32 unlabeled images and about ten minutes of single-core computation.
Keywords: Model compression; Mixed-precision quantization; Structured pruning; Real-time object detection; Edge inference; Latency modeling

1 INTRODUCTION

Real-time visual perception is a core building block of mobile robots, smart cameras, driver-assistance units and industrial inspection systems. Single-stage detectors such as YOLOv8 [1], YOLOv7 and YOLOv6 [2-3], as well as real-time transformers such as RT-DETR [4], reach high accuracy at video rate on desktop GPUs, yet many deployed perception nodes run on low-power CPUs that offer a handful of cores, a few gigabytes of memory and no accelerator. On such platforms even the smallest detectors miss typical frame-rate budgets, and model compression becomes a prerequisite rather than an option. The two most deployable compression families are integer quantization [5-8], which replaces 32-bit floating point arithmetic by 8-bit integer kernels, and structured (channel) pruning [9-10], which removes whole filters and therefore shrinks dense computation without sparse kernels; both are frequently combined [11].

Applying these techniques uniformly, i.e., one precision and one pruning ratio for every layer, leaves accuracy on the table because layers differ widely in how much they contribute to the final prediction. Mixed-precision methods such as HAQ [12], HAWQ [13], HAWQ-V3 and ZeroQ [14-15] allocate bit-widths per layer, and latency-aware pruning methods such as HALP and SPDY [16-17] allocate pruning ratios under a latency target. Three issues limit their effectiveness for CPU-based real-time perception. First, each method optimizes a single compression dimension, although precision and width interact through the same latency budget. Second, cost models are usually additive per-layer lookup tables or FLOP/bit-operation counts. On CPU runtimes this assumption breaks: keeping one layer in floating point inside an otherwise quantized graph forces dequantization of its inputs, quantization of its outputs and a floating-point activation, so the latency of a mixed-precision model depends on how precisions are arranged, not only on how many layers are quantized. Third, sensitivities are typically measured around the full-precision model with task-agnostic proxies, whereas the deployed model lives near the quantized operating point and only a subset of the detector outputs matters to the application.

This paper presents ACORN (Accuracy-aware COmpression for Real-time perceptioN), a post-training framework that jointly allocates per-component pruning ratios and numerical precision to a pretrained detector under explicit latency and memory budgets. ACORN requires 32 unlabeled images, no gradient computation and no fine-tuning, and its output is a standard ONNX model executed by an off-the-shelf runtime. The main contributions are as follows.

1) A joint allocation of channel-pruning ratio and precision per compression unit, formulated as an integer linear program (ILP) with latency and memory constraints and solved exactly with an off-the-shelf MILP solver.

2) A deployment-anchored, task-aware sensitivity estimator that scores every compression option as a perturbation of the uniformly quantized deployment model, using a confidence-weighted output discrepancy restricted to the task-relevant categories.

© By the Author(s) 2022, under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0).

3) A conversion-aware latency model obtained by profiling the runtime-optimized graph, which adds pairwise precision-switch (quantize/dequantize) costs to an additive lookup table, together with a training-free channel pruning operator based on ridge least-squares reconstruction of the consuming convolution.
4) A comprehensive evaluation on YOLOv8n that reports accuracy, measured single-core latency, model size and runtime memory against quantization, mixed-precision and pruning baselines from recent literature.

2 RELATED WORK

2.1 Quantization and Mixed Precision

Integer-arithmetic-only inferenceand the practical recipes collected in underpin most deployed post-training quantization (PTQ) pipelines [6-8] ; the survey in covers the wider design space [5]. Rounding and reconstruction techniques such as AdaRound [19], BRECQ [20], QDrop and PD-Quant reduce the error of low-bit PTQ by optimizing rounding or block outputs [21-22]; PD-Quant in particular argues that the prediction difference is a better objective than local feature distances, a view that ACORN shares at the level of allocation. Mixed-precision allocation chooses a precision per layer: HAQ uses reinforcement learning with hardware feedback, HAWQ and HAWQ-V3 rank layers with Hessian information and solve an ILP, and ZeroQ measures the output KL divergence of single-layer quantization. These estimators perturb one layer of the full-precision network at a time and treat latency as a sum of per-layer costs [14-15].

2.2 Structured Pruning

Filter pruning by weight magnitude is the simplest structured criterion [9]. Channel pruning and ThiNet select channels using the statistics of the next layer and reconstruct its output by least squares [23-24], which makes moderate pruning possible with little or no retraining. DepGraph generalizes structural pruning to arbitrary architectures by grouping coupled parameters through a dependency graph and ranking groups by their norm [10]. HALP formulates latency-constrained pruning as a knapsack problem over a latency lookup table [16], SPDY searches layer-wise sparsity with speedup guarantees [17], and Optimal Brain Compression performs post-training pruning and quantization with second-order layer-wise solvers [18].

2.3 Joint and Hardware-Aware Compression

Deep Compression applies pruning and quantization sequentially [11]. AMC searches pruning ratios with reinforcement learning [25], APQ jointly searches architecture, pruning and quantization with an accuracy predictor [26], and DJPQ learns pruning and bit-widths differentiably [27]; these methods require training or extensive search. Platform-aware design methods such as ChamNet rely on latency lookup tables and predictors [28]. ACORN differs from these lines in being training-free, in optimizing pruning and precision jointly over a single exact ILP, in anchoring sensitivity at the quantized deployment point, and in modeling the precision-switch overheads that appear on CPU inference runtimes.

3 METHOD

3.1 Compression Units and Problem Formulation

ACORN operates on the inference graph of a pretrained detector. A *compression unit u* is either a convolution module (a convolution followed by its SiLU activation) or a *prunable chain*, i.e., a pair of modules A→B in which the activation of A is consumed exclusively by the convolution B. Chains are detected automatically from the graph; overlapping candidates are resolved greedily by the number of FLOPs they control. In YOLOv8n, 21 chains cover 81% of the convolutional FLOPs, and together with 21 single modules they form 42 units. The box decoder (distribution-focal integral and coordinate decoding) is kept in FP32, and glue operators (Concat, Split, Add, MaxPool, Resize) are quantized whenever the model is quantized.
Each single module chooses a precision $q \in \{\text{INT8}, \text{FP32}\}$; each chain chooses a pair (r, q), where $r \in R = \{0, 0.25, 0.5, 0.75\}$ is the fraction of internal channels removed. With binary variables $x_{u,o}$ that select option o of unit u, ACORN solves

$$\min_x \Sigma_u \Sigma_{o \in O_\{u\}} S_u(o)\, x_{u,o} \quad \text{s.t.} \quad \hat{T}(x) \le T, \quad \hat{M}(x) \le M, \quad \Sigma_o x_{u,o} = 1 \ \forall u, \tag{1}$$

where $S_u(o)$ is the accuracy cost of the option, $\hat{T}$ the predicted latency, $\hat{M}$ the predicted weight memory, and T, M the budgets. The three ingredients are described below.

3.2 Training-Free Structured Pruning

Removing output channels of A in a chain changes only the input channels of B; every tensor outside the chain keeps its shape, so chains can be compressed independently and the external interface of the network is preserved. For a chain with C internal channels, ACORN ranks channel c by its contribution to the consumer,

$$s_c = \|W_B[:, c]\|_F \cdot \left(E[a_c^2] \right)^{1/2}, \tag{2}$$

where a_c is the activation of channel c on the calibration images, and keeps the top $\lceil (1-r)C \rceil$ channels rounded to a multiple of 4 or 8 for SIMD efficiency. The consumer convolution is then re-fitted. Let Φ_K contain the im2col patches of the retained channels at $P = 16{,}000$ randomly sampled output positions and Y the pre-activation outputs of B in the original FP32 model at the same positions. The new weights and bias follow from ridge least squares,

$$\hat{W}_B = (\tilde{\Phi}_K^T \tilde{\Phi}_K + \lambda I)^{-1} \tilde{\Phi}_K^T Y, \quad \lambda = 10^{-3} \operatorname{tr}(\Phi_K^T \Phi_K)/d, \tag{3}$$

with $\tilde{\Phi}_K = [\Phi_K, 1]$ and d the patch dimension. When several chains are pruned, they are processed in topological order and Φ_K is taken from the already compressed network while Y comes from the original one, so each reconstruction also compensates the error accumulated upstream. The cost is one forward pass and one $d \times d$ solve per chain.

3.3 Deployment-Anchored, Task-Aware Sensitivity

Let F_0 denote the deployment anchor, i.e., the network with every unit quantized to INT8 and no pruning, and let $F_0 \oplus (u \to o)$ be the anchor in which only unit u takes option o. For a detector with anchor-wise class probabilities $p_{a,c}$ and boxes b_a, the discrepancy between a reference output (p, b) of the FP32 model and a compressed output $(\hat{p}, \hat{b})$ on image x is

$$D(x) = \Sigma_a w_a \left[\Sigma_{c \in C_\{T\}} |p_{a,c} - \hat{p}_{a,c}| + 1 - \mathrm{IoU}(b_a, \hat{b}_a) \right] / \Sigma_a \max_{c \in C_\{T\}} p_{a,c}, \tag{4}$$

with $w_a = \max(\max_{c \in C_\{T\}} p_{a,c}, \max_{c \in C_\{T\}} \hat{p}_{a,c})$ and C_T the set of task-relevant categories. The weight focuses the measure on anchors that are confident in either model, which are exactly those that survive non-maximum suppression and determine average precision; including the compressed confidence also penalizes spurious detections. Restricting the sum to C_T discards changes in categories the application never reports. The option cost is the change of the expected discrepancy relative to the anchor,

$$S_u(o) = E_x[D(x; F_0 \oplus (u \to o))] - E_x[D(x; F_0)]. \tag{5}$$

Keeping a unit in FP32 yields a negative cost (an accuracy gain), pruning a positive one. For chains, the joint options (r, INT8) are measured directly, which captures the interaction between reconstruction and the activation ranges seen by the quantizer; the FP32 variants are composed as $S_u(r, \text{FP32}) = S_u(r, \text{INT8}) + S_u(0, \text{FP32})$. For YOLOv8n this requires $42 + 63 = 105$ quantized variants evaluated on 16 unlabeled images. Linearizing the accuracy loss at F_0 instead of at the FP32 model is the key design choice: under budgets at or below the INT8 latency almost all units remain INT8, so the solution lies close to the anchor, and the measured costs already include the error accumulated through the other quantized layers.

3.4 Conversion-Aware Latency and Memory Model

Latency is measured on the target device after the runtime has optimized the graph, because fusions such as DequantizeLinear–Conv–QuantizeLinear → QLinearConv and layout transformations change the operator set substantially. Per-node kernel times are profiled for $2|R|$ configurations (uniform ratio r with all units INT8, and with all units FP32 inside an INT8 graph) plus the pure FP32 graph, in interleaved rounds to suppress drift. Kernel and activation nodes are attributed to units by name; conversion nodes (quantize, dequantize, layout reorders) are attributed to the FP32 unit they serve. This yields an additive table $t_u(o)$. Because conversions between two directly connected units appear only when their precisions differ, ACORN adds pairwise terms on the set E of direct unit-to-unit edges,

$$\hat{T} = t_0 + \Sigma_u t_u(o_u) + \Sigma_{(u,v) \in E} [c^{DQ}_{uv} d^1_{uv} + c^Q_{uv} d^2_{uv}], \tag{6}$$

where $c^{DQ}_{uv} = \alpha_{DQ} n_{uv}$ and $c^Q_{uv} = \alpha_Q n_{uv}$ scale with the element count n_{uv} of the connecting tensor, and the per-element coefficients α are fitted on the profiled conversion nodes. Weight memory is modeled exactly from parameter counts: 4 bytes per FP32 weight, and 1 byte per INT8 weight plus a per-channel scale, zero point and 32-bit bias.

3.5 Exact Allocation by Integer Linear Programming

With $\pi_u = \Sigma_{o \in O\^\{FPu\}} x_{u,o}$ indicating an FP32 unit, the switch indicators of (6) are linearized as

$$d^1_{uv} \geq \pi_v - \pi_u, \quad d^2_{uv} \geq \pi_u - \pi_v, \quad d^1_{uv}, d^2_{uv} \in \{0, 1\}, \tag{7}$$

which is exact because the switch variables appear only in the latency constraint with non-negative coefficients. Problem (1) then becomes a small ILP (fewer than 300 binary variables for YOLOv8n) that the HiGHS solver solves to optimality. The selected allocation is materialized by sequential pruning with reconstruction, MinMax range estimation on the compressed network, and QDQ quantization of all INT8 units, producing a single ONNX file. Algorithm 1 summarizes the procedure.

Algorithm 1 (ACORN). Input: FP32 model, 16 calibration and 16 sensitivity images, budgets T, M. (i) Discover units and chains; estimate MinMax ranges; build the anchor F_0. (ii) For every unit-option pair, build $F_0 \oplus (u \to o)$ (pruning with reconstruction where $r > 0$) and record $S_u(o)$ by (5). (iii) Profile the optimized graph and fit $t_u(o)$, t_0 and α. (iv) Solve the ILP (1), (6), (7). (v) Materialize, calibrate and export the compressed model (Figure 1).

4 EXPERIMENTAL SETUP

Model and task. We compress YOLOv8n (3.15 M parameters [1], 8.7 GFLOPs at 640×640) pretrained on MS COCO and exported to ONNX (opset 17) [30]. The perception task is the 20-category PASCAL VOC2007 detection problem [29]; every VOC category corresponds to a COCO category, so the detector is deployed with a task head that reads the

20 corresponding class scores. Accuracy is measured on a fixed subset of 1,238 test images (every fourth image of the official test list) with the VOC devkit matching rule (difficult objects ignored) and reported as mAP@0.5 and mAP@[.5:.95] with all-point interpolation; inputs are letterboxed to 640×640, the confidence threshold is 0.001, NMS uses IoU 0.7 and at most 300 detections are kept.

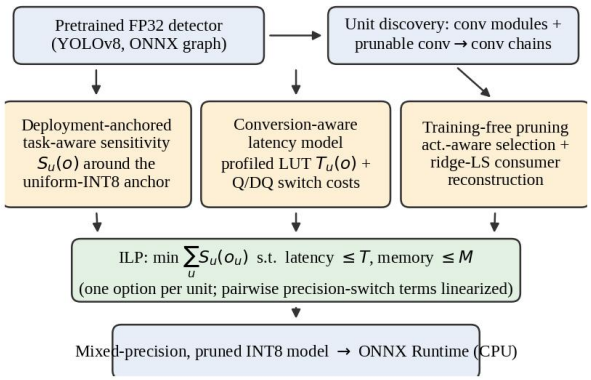


Figure 1 Overview of ACORN
Sensitivities, latency costs and the pruning operator feed one exact integer program whose solution is exported as a single mixed-precision, pruned ONNX model.

Compression data. Sixteen calibration images (range estimation and reconstruction) and sixteen sensitivity images are drawn from VOC2007 trainval, disjoint from the test images; labels are never used. *Runtime and hardware.* All models run in ONNX Runtime 1.24 (CPU execution provider, one intra-op thread, batch size 1) on one core of an Intel Xeon processor at 2.1 GHz with AVX-512 VNNI and 4 GB of memory. Latency is the median of two interleaved rounds of 30 inferences after warm-up (network only). Size is the ONNX file size, and memory is the increase of the peak resident set size of an inference process. Quantization uses the QDQ format with per-channel symmetric INT8 weights and per-tensor asymmetric UINT8 activations over the full graph.
Budgets and baselines. The latency budgets are T_a, the predicted latency of uniform INT8 (iso-latency), T_b and T_c, the predicted latencies of uniform 25% and 50% channel pruning on top of INT8, and $T_{mp} = 1.15T_a$ for precision-only allocation. We compare with INT8 PTQ using MinMax and 99.999-percentile calibration [6-8]; ZeroQ-MP, which allocates precision with the KL sensitivity of ZeroQ through the same ILP and latency model [15]; DepGraph-U25, uniform 25% pruning with the group-norm criterion of DepGraph [10]; and HALP-K, a latency-saliency knapsack in the spirit of HALP with layer-normalized group saliency at T_b [16]. All pruning baselines use the same least-squares reconstruction and INT8 quantization as ACORN, and no method uses fine-tuning, so differences isolate the allocation decisions.

5 RESULTS AND DISCUSSION

5.1 Main Comparison

Table 1 summarizes accuracy, latency, model size and runtime memory. All latencies in Table 1 were re-measured in one interleaved benchmarking session; the allocation budgets were fixed earlier from the profiling session and are therefore stated relative to uniform INT8, with $T_b \approx 0.94T_a$ and $T_c \approx 0.86T_a$. Full-graph INT8 quantization with MinMax ranges is a strong baseline: it accelerates the detector by 2.28× and shrinks it from 12.85 MB to 3.66 MB at a cost of 0.77 points of mAP@[.5:.95]. Percentile calibration lowers the accuracy to 61.02%, suggesting that the large activations removed by clipping carry information that the detection head relies on.
At the iso-latency budget T_a, ACORN keeps the stem chain (0) and the module 2.cv1 in FP32 and pays for them by removing 75% of the internal channels of cls-P3.0. The resulting model runs at the measured latency of uniform INT8 (41.8 ms versus 41.8 ms) with a slightly smaller file, recovers 0.70 points of mAP@[.5:.95] and reaches 83.98% mAP@0.5, i.e., the accuracy level of the FP32 model (63.54% and 83.74%) at 2.28× its speed. At T_b, ACORN keeps a single unit (box-P5.2) in FP32 and prunes box-P3.0 and cls-P3.0 by 75%; the model is 9.0% faster than uniform INT8 and still 0.40 points more accurate.
The pruning baselines show how much the allocation matters in the training-free regime. With identical reconstruction and quantization, uniform 25% pruning with the DepGraph group-norm criterion collapses to 30.33%, because every chain, including the most sensitive ones, loses a quarter of its width. The HALP-style knapsack distributes pruning according to latency savings and layer-normalized weight saliency and reaches 52.54% at 38.8 ms, 10.6 points below ACORN at a comparable latency (38.1 ms). Weight saliency does not reveal which channels the detection outputs depend on, whereas the anchored discrepancy measures the effect on the final predictions directly.
Precision-only allocation needs a larger budget. With 15% more predicted latency (T_{mp}), ZeroQ-MP and ACORN-MP keep 4 and 9 units in FP32 and reach 63.01% and 63.00% at 47.7 and 47.1 ms, respectively. The joint allocation at T_a is both faster and more accurate (63.47% at 41.8 ms), because pruning an insensitive chain releases more latency than the

additional FP32 units consume. At the aggressive budget T_c (2.79× speedup over FP32), ten chains are pruned by 25–75% and the accuracy decreases to 37.68%. This point marks the knee of the training-free trade-off in Figure 2; budgets of this kind call for a short fine-tuning stage, for which the ACORN allocation provides the initialization.

Table 1 Accuracy (%) and Efficiency of Compressed YOLOv8n on One CPU Core (mAP = mAP@[.5:.95])

Method	mAP_{50}	mAP	Lat. (ms)	Speedup	Size (MB)	Mem. (MB)
FP32 (reference)	83.7	63.5	95.6	1.00×	12.85	96.3
INT8-MinMax [6]	83.1	62.8	41.8	2.28×	3.66	48.3
INT8-Percentile [8]	82.3	61.0	40.2	2.38×	3.66	48.3
ZeroQ-MP [15] @T_{mp}	83.4	63.0	47.7	2.00×	3.73	68.5
ACORN-MP @T_{mp}	**83.3**	**63.0**	47.1	2.03×	4.64	63.9
DepGraph-U25 [10]	48.0	30.3	38.9	2.46×	3.07	51.2
HALP-K [16] @T_b	72.6	52.5	38.8	2.46×	3.60	44.6
ACORN @T_a	**84.0**	**63.5**	41.8	2.28×	3.58	53.8
ACORN @T_b	**83.6**	**63.2**	38.1	2.51×	3.53	50.6
ACORN @T_c	**54.3**	**37.7**	34.2	2.79×	3.26	53.2

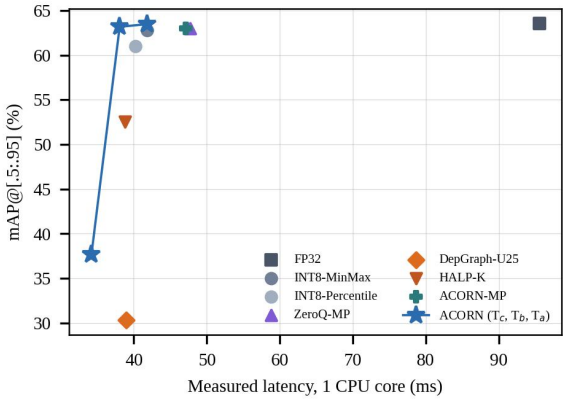


Figure 2 Accuracy–Latency Trade-Off of Compressed YOLOv8n on One CPU Core (VOC2007 Test Subset)
The ACORN curve connects the budgets T_c, T_b and T_a.

5.2 Where Does the Latency Go?

Figure 3 decomposes the profiled latency into compression units. In the profiling session the pure FP32 graph takes 134.2 ms and the uniform INT8 anchor 65.0 ms. Two effects shape the allocation. First, the INT8 gain is uneven across units: high-resolution layers with few channels gain least, whereas wide 3×3 convolutions such as those of the P3 head branches gain most. Second, a unit executed in FP32 inside an INT8 graph pays for dequantizing its input and quantizing its output. With all units in FP32 and only the glue operators in INT8, the graph takes 138.3 ms, so the conversions consume the savings of the INT8 glue operators. The fitted conversion costs are 0.35 ms for quantization and 0.45 ms for dequantization per million elements, and 18 direct unit-to-unit edges of YOLOv8n carry such pairwise terms. The same effect explains why quantizing only the convolutions, leaving SiLU and the glue operators in FP32, is far less effective: the conv-only INT8 model needs 57.1 ms, against 41.8 ms for full-graph INT8.

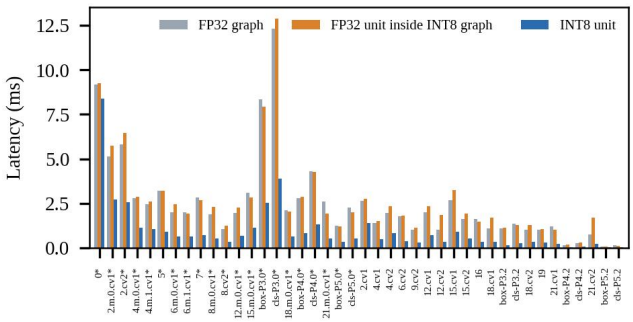


Figure 3 Per-Unit Latency Obtained by Profiling the Optimized ONNX Runtime Graph
Units in the pure FP32 graph, FP32 units embedded in an INT8 graph (including conversion overhead), and INT8 units. Asterisks mark prunable chains.

5.3 Accuracy of the Latency Model

Table 2 and Figure 4 compare the conversion-aware model with an isolated per-layer LUT that uses the cost of each unit in the pure FP32 graph and ignores precision switches. The test set consists of 11 allocations (ten random allocations with 10–50% FP32 units and random pruning ratios, plus the INT8 anchor). Predictions come from the profiling session and are compared with wall-clock measurements; the affine column removes the global offset between the two sessions. The conversion-aware model reaches $R^2 = 0.936$ and 3.3% mean absolute percentage error after affine calibration, compared with $R^2 = 0.938$ and 3.1% for the isolated LUT. On this processor both models rank the candidate allocations similarly, and the conversion terms mainly correct the absolute cost of FP32 units.

Table 2 Accuracy of Latency Models on Random Allocations

Latency model	R^2	MAPE (raw)	MAPE (affine)
Conversion-aware (ours)	0.936	5.2%	3.3%
Isolated per-layer LUT	0.938	7.4%	3.1%

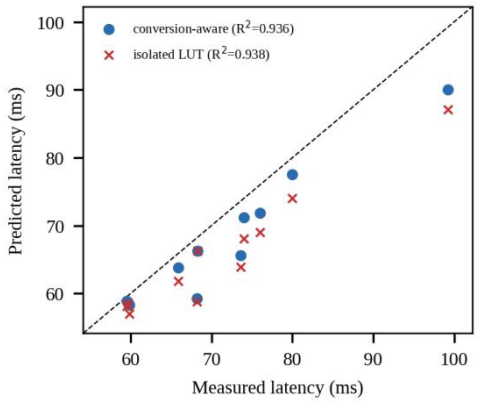


Figure 4 Predicted versus Measured Latency of Random Mixed-Precision, Pruned Allocations of YOLOv8n

5.4 Sensitivity Structure and Allocations

Figure 5 shows the anchored sensitivities. The accuracy gain of FP32 execution is concentrated in a few units, led by 0, 2.cv1 and 7, while most units contribute little once the rest of the network is quantized. Pruning costs differ by a factor of 49 across chains at the same 50% ratio and grow quickly with the ratio for the sensitive chains. These two observations explain the allocations in Figure 6: ACORN removes width where the anchored cost is small and spends the released latency on the few units whose FP32 execution matters, and when the budget tightens to T_c it spreads pruning over more chains instead of deepening the cuts in sensitive ones.

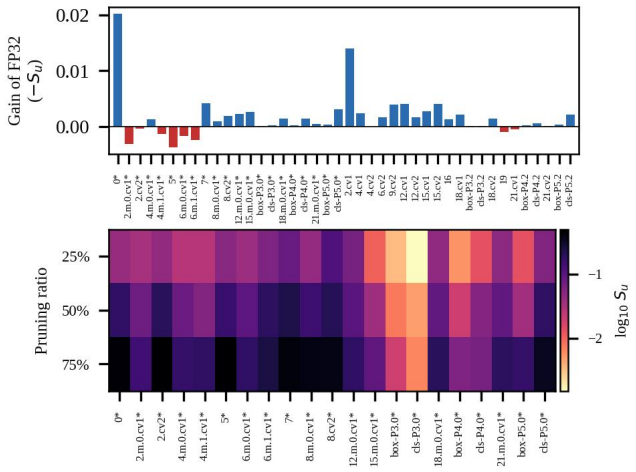


Figure 5 Deployment-Anchored Sensitivities of YOLOv8n
Top: accuracy gain of keeping each unit in FP32 inside the INT8 anchor. Bottom: cost of pruning each chain by 25%, 50% and 75% (log scale).

5.5 Ablation Study

Table 3 isolates the design choices at T_b. Replacing the task-aware weighting by a uniform average over all anchors and all 80 classes is catastrophic: the uniform discrepancy is dominated by background anchors and underrates the damage of aggressively pruning cls-P4.0 and cls-P5.0, and the resulting allocation yields only 2.93% mAP@[.5:.95]. Restricting

the discrepancy to the deployed categories and weighting it by confidence is therefore the most important ingredient of the estimator.

The remaining variants reach the same decision for YOLOv8n. Isolated FP32-anchored sensitivities and the isolated LUT return exactly the allocation of the full method, and the pruning-only option space reaches the same accuracy (63.17%) because ACORN keeps just one unit in FP32 at this budget. The joint option space matters most at T_a: the FP32 units selected there are paid for by pruning, and the model gains 0.70 points at unchanged latency, whereas the precision-only allocations of Table 1 need a 15% larger budget to approach this accuracy.

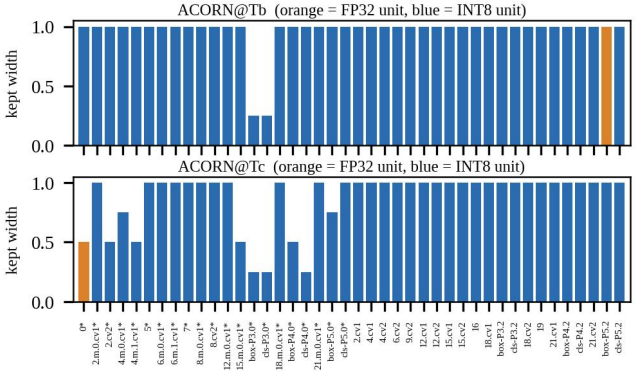


Figure 6 Allocations Selected by ACORN at T_b and T_c: Retained Width (1−r) and Precision of Each Unit

Table 3 Ablation of ACORN at Budget T_b

Variant	mAP_{50}	mAP	Lat. (ms)	FP32 units	Mean r
ACORN (full)	83.6	63.2	38.1	1	0.071
w/o deployment anchoring	83.6	63.2	37.4	1	0.071
w/o task-aware weighting	4.6	2.9	38.1	2	0.155
pruning options only	83.7	63.2	37.2	0	0.071
isolated LUT (no switch cost)	83.6	63.2	37.4	1	0.071

5.6 Memory Footprint and Compression Cost

Full-graph INT8 reduces the ONNX file from 12.85 MB to 3.66 MB and the runtime memory of the inference process from 96.3 MB to 48.3 MB. The ACORN models occupy 3.26–3.58 MB. Their runtime memory (50.6–53.8 MB) is slightly above that of uniform INT8 because FP32 units keep floating-point activation buffers, and it stays near half of the FP32 footprint; the memory constraint of (1) bounds this quantity directly when the target device is memory-limited. The complete allocation procedure, consisting of 105 anchored sensitivity evaluations (8.6 min) and latency profiling with validation (1.7 min), runs on the same single CPU core as the deployment, so the compression can be repeated on the target device whenever the budget or the runtime changes.

6 CONCLUSION

This paper presented ACORN, a post-training framework for accuracy-aware compression of real-time perception models under latency and memory constraints. ACORN measures the accuracy cost of every pruning and precision option around the quantized deployment point with a task-aware discrepancy, models the latency of mixed-precision graphs from the runtime-optimized operator profile including precision-switch costs, and solves the joint allocation exactly. On YOLOv8n and a single CPU core, the resulting standard ONNX models recover the FP32 detection accuracy at the speed of uniform INT8 quantization and clearly outperform uniform and knapsack-based structured pruning at higher speedups, while also reducing model size and runtime memory.

Future work will extend the allocation space to low-bit weights, input resolution and multi-model perception pipelines, transfer the profiling procedure to embedded GPUs and NPUs, and couple the allocation with short distillation-based fine-tuning for aggressive budgets.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Jocher G, Chaurasia A, Qiu J. Ultralytics YOLOv8. Version 8.0.0, 2023. https://github.com/ultralytics/ultralytics.

[2] Wang C Y, Bochkovskiy A, Liao H Y M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023: 7464-7475.

[3] Li C, Wang Z, Yang C, et al. YOLOv6: A single-stage object detection framework for industrial applications. arXiv:2209.02976, 2022.
[4] Lv W, Xu S, Zhao Y, et al. DETRs beat YOLOs on real-time object detection. arXiv:2304.08069, 2023.
[5] Gholami A, Kim S, Dong Z, et al. A survey of quantization methods for efficient neural network inference. In: Low-Power Computer Vision. Boca Raton, FL, USA: CRC Press, 2022: 291-326.
[6] Jacob B, Kligys S, Chen B, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018: 2704-2713.
[7] Krishnamoorthi R. Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv:1806.08342, 2018.
[8] Nagel M, Fournarakis M, Amjad R A, et al. A white paper on neural network quantization. arXiv:2106.08295, 2021.
[9] Li H, Kadav A, Durdanovic I, et al. Pruning filters for efficient ConvNets. In: Proceedings of the International Conference on Learning Representations (ICLR), 2017.
[10] Fang G, Ma X, Song M, et al. DepGraph: Towards any structural pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023: 16091-16101.
[11] Han S, Mao H, Dally W J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In: Proceedings of the International Conference on Learning Representations (ICLR), 2016.
[12] Wang K, Liu Z, Lin Y, et al. HAQ: Hardware-aware automated quantization with mixed precision. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019: 8612-8620.
[13] Dong Z, Yao Z, Gholami A, et al. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019: 293-302.
[14] Yao Z, Dong Z, Zheng Z, et al. HAWQ-V3: Dyadic neural network quantization. In: Proceedings of the International Conference on Machine Learning (ICML), 2021: 11875-11886.
[15] Cai Y, Yao Z, Dong Z, et al. ZeroQ: A novel zero shot quantization framework. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020: 13169-13178.
[16] Shen M, Yin H, Molchanov P, et al. Structural pruning via latency-saliency knapsack. In: Advances in Neural Information Processing Systems (NeurIPS), 2022.
[17] Frantar E, Alistarh D. SPDY: Accurate pruning with speedup guarantees. In: Proceedings of the International Conference on Machine Learning (ICML), 2022: 6726-6743.
[18] Frantar E, Singh S P, Alistarh D. Optimal brain compression: A framework for accurate post-training quantization and pruning. In: Advances in Neural Information Processing Systems (NeurIPS), 2022.
[19] Nagel M, Amjad R A, van Baalen M, et al. Up or down? Adaptive rounding for post-training quantization. In: Proceedings of the International Conference on Machine Learning (ICML), 2020: 7197-7206.
[20] Li Y, Gong R, Tan X, et al. BRECQ: Pushing the limit of post-training quantization by block reconstruction. In: Proceedings of the International Conference on Learning Representations (ICLR), 2021.
[21] Wei X, Gong R, Li Y, et al. QDrop: Randomly dropping quantization for extremely low-bit post-training quantization. In: Proceedings of the International Conference on Learning Representations (ICLR), 2022.
[22] Liu J, Niu L, Yuan Z, et al. PD-Quant: Post-training quantization based on prediction difference metric. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023: 24427-24437.
[23] He Y, Zhang X, Sun J. Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017: 1389-1397.
[24] Luo J H, Wu J, Lin W. ThiNet: A filter level pruning method for deep neural network compression. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017: 5058-5066.
[25] He Y, Lin J, Liu Z, et al. AMC: AutoML for model compression and acceleration on mobile devices. In: Proceedings of the European Conference on Computer Vision (ECCV), 2018: 784-800.
[26] Wang T, Wang K, Cai H, et al. APQ: Joint search for network architecture, pruning and quantization policy. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020: 2078-2087.
[27] Wang Y, Lu Y, Blankevoort T. Differentiable joint pruning and quantization for hardware efficiency. In: Proceedings of the European Conference on Computer Vision (ECCV), 2020: 259-277.
[28] Dai X, Zhang P, Wu B, et al. ChamNet: Towards efficient network design through platform-aware model adaptation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019: 11398-11407.
[29] Everingham M, Van Gool L, Williams C K I, et al. The PASCAL visual object classes (VOC) challenge. International Journal of Computer Vision, 2010, 88(2): 303-338.
[30] Lin T Y, Maire M, Belongie S, et al. Microsoft COCO: Common objects in context. In: Proceedings of the European Conference on Computer Vision (ECCV), 2014: 740-755.