Self-Compressing Vision Tower for Efficient Dense Prediction Tasks

Aditya Shourya[1][0009−0003−4094−5455], Guangzhi Tang[1][0000−0002−0204−9225], and Chang Sun[1,2][0000−0001−8325−8848]

[1] Department of Advanced Computing Sciences, Maastricht University, Netherlands
[2] Institute of Data Science, Maastricht University, Netherlands
a.shourya@student.maastrichtuniversity.nl,
{guangzhi.tang, chang.sun}@maastrichtuniversity.nl

Abstract. Dense prediction tasks such as classification, segmentation, and optical flow require models that deliver high accuracy while maintaining sufficient throughput for practical applications on mobile or portable computing devices. However, most state-of-the-art architectures rely on deep sequential operations that are computationally expensive and challenging to execute on consumer-grade parallel hardware; this often leads to reduced inference speed or degraded accuracy, thereby limiting their applicability in real-time and edge scenarios. To address this challenge, we propose a novel, self-compressing vision architecture that applies structured pruning and quantization across key modules: convolutional layers, transposed convolutions, and linear attention in proportion to their parallel-time computational cost. By selectively reducing precision and pruning tensors in less critical layers, our approach achieves significant model compression. We evaluate our method on fine-grained classification (CUB-200-2011, Country211), semantic segmentation (ADE20K), and optical flow (HD1K). Our model matches the accuracy of state-of-the-art baselines (Efficient VIT) at full precision (FP32) and surpasses them under lower-precision settings, achieves reduced storage and higher throughput, all while maintaining similar training time. Finally, we highlight that compression serves not only as a mechanism for reducing model size but also as a basis for investigating the relationship between model depth and overall performance during inference.
Project at: https://github.com/adishourya/SelfcompressingDepthWiseAttn

Keywords: Dense Prediction · Model Compression · Structured Pruning · Quantization · Efficient Vision Models

1 Introduction

Dense prediction remains a critical task in computer vision, and the efficacy of a vision model is commonly benchmarked by its performance in both accuracy and latency on large-scale dense prediction datasets [6]. Models with low computational demands are essential for real-time applications, including

autonomous drones, on-device image processing, and computational photography [26]. In specific domains such as sports or wildlife photography, high inference throughput is often as critical as high accuracy [40].

Despite this need, many state-of-the-art vision architectures [5] remain too computationally expensive for deployment on consumer-grade devices. A key insight from recent works [9, 14] shows the possibility of over-parameterization in deep networks. The findings suggest that the latter layers of deep sequential models often contribute marginally less to final accuracy than earlier ones. This challenges the necessity of the high computational cost inherent in deep architectures and motivates the design for more efficient models. However, most post-training methods [16, 17] that are applied to the state of the art vision models to meet device compute capability constraints, often incur significant drop in accuracy, particularly in dense prediction tasks where fine-grained spatial details are critical.

A key hypothesis we examine in this work is the need for uniform allocation of model precision and compute cost across model depth. In non-self-compressing models [22], [36], weight precision and compute budgets are assigned identically to all layers regardless of their relative contribution to final performance. This uniform treatment can result in over-parameterization and inefficient use of compute resources.

Motivated by previous work, we present a fully compressible vision backbone that matches the accuracy of the current state-of-the-art model EfficientViT [5] at full precision and surpasses it under low-precision settings. Our architecture integrates structured pruning and quantization-aware training to dynamically adjust model precision and computational cost across different compute modules, including convolutional filters and attention weights. This non-uniform compression strategy yields substantial reductions in storage and compute requirements with minimal impact on accuracy. We validate the robustness and generalizability of proposed approach across multiple dense prediction tasks such as a fine-grained classification (CUB-200-2011 [35], Country211 [28]), semantic segmentation (ADE20K [41]), and optical flow (HD1K [20]). We further show that the compression process can serve as an analysis tool for identifying and prioritizing critical model components, informing the design of more efficient architectures.

2 Related Study

Our methodology builds upon recent advancements in efficient vision towers and quantization approaches, drawing insights from prior work. Below, we summarize key contributions from related works that have informed our approach.

Efficient ViT [5] carefully selects computational operations to maximize accuracy while sustaining high inference throughput. The design philosophy delegates the computation of local inductive biases to lightweight convolutional kernels and global feature aggregation to linear attention mechanisms, avoiding the quadratic

complexity of full softmax attention [19]. The local processing module adopts inverted bottleneck convolutions [30], which first expand the channel dimension, apply a depthwise convolution for spatial processing, and then project the features back to a lower dimension. This structure enables efficient feature learning in a higher-dimensional space while keeping computational costs low.

Our work adopts a similar architectural philosophy to achieve high throughput. However, unlike EfficientViT, which focuses primarily on operator efficiency, our approach also integrates compression techniques such as pruning and quantization directly into these modules to further improve throughput, reduce model size, and maintain accuracy.

DiffQ[11] introduces a differentiable quantization framework that avoids gradient approximations such as the Straight-Through Estimator (STE). It models quantization as the addition of pseudo-quantization noise during the forward pass, enabling direct gradient flow through the quantization step. The method jointly learns both full-precision weights and their bit-depths, with a regularization term that penalizes model size, controlled by a single hyperparameter. This allows automatic discovery of mixed-precision strategies, allocating higher precision to more sensitive parameters.

While DiffQ achieves strong compression rates across vision, language, and audio tasks, it incurs additional training overhead from generating noise for every group of weights the noise parameter is tied to, and requires careful tuning of its trade-off hyperparameter. Moreover, it focuses solely on minimizing storage rather than directly optimizing for latency. Inspired by DiffQ's bit-depth regularization, we adopt a similar precision-penalizing strategy but use STE-based gradient approximation to avoid the computational overhead of noise injection.

Self-Compressing Convolutions (SCC) [8] removes the need for noise injection by directly parameterizing bit-depth as a trainable continuous variable. A tempered sigmoid gate allows smooth reduction of bit-depths to zero, effectively pruning weights or entire channels (modes of tensor) during training. Unlike DiffQ's fine-grained grouping, SCC typically operates on coarser structural units, improving stability and reducing catastrophic forgetting.

Although originally applied to only convolutional layers, SCC's principles can extend to other operator types. In our work, we generalize this approach to compress all major components of the vision tower, including attention mechanisms, transposed convolutions, and linear projections, enabling a fully compressible architecture.

3 Architecture

The architecture design of this work was inspired by the recent vision models [5] and [8]. The proposed architecture follows a decoder-only transformer structure (Figure 1). The input image is first projected into a latent sequence of embeddings,

where each embedding dimension corresponds to a learnable convolution kernel. Each transformer block is then implemented as a sequential composition of convolutional and transpose convolution operations [39, 15, 29] followed by a linear attention module to capture global receptive fields as shown in Figure 2.

Quantization is applied to the parameter matrices of all the aforementioned compute modules. The compression penalty for each module is designed to be directly proportional to its parallel-time arithmetic intensity, with the expectation that this scaling induces an inverse relationship with the measured throughput on a generic parallel accelerator. This formulation explicitly favors operations

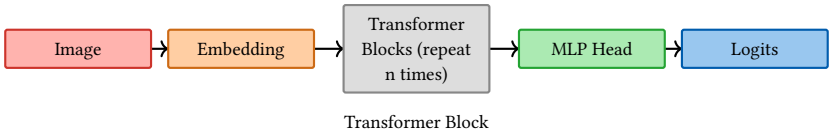


Fig. 1: ***Overall Architecture**: The input image is processed through a series of transformer blocks, each consisting of convolutional operations (Local Module) and linear attention (Context Module), followed by an MLP head for dense prediction tasks.*

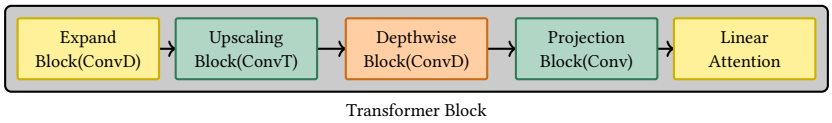
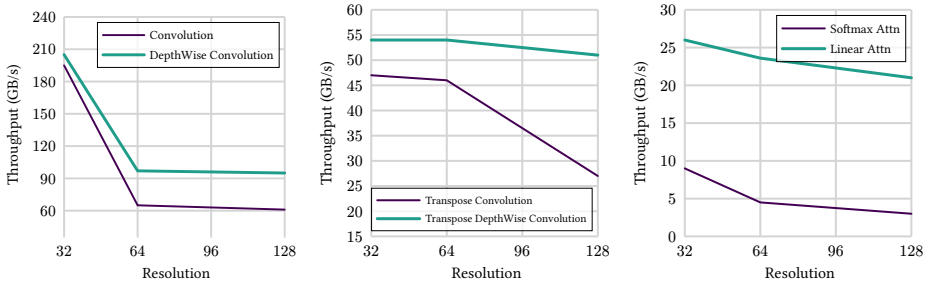


Fig. 2: ***Transformer Block Detail**: Each transformer block (in Figure 1) consists of a modified Inverted Bottleneck Convolution (expansion stage using depthwise convolution (ConvD), an upscaling stage using transposed convolution (ConvT), and a depthwise convolution before a projection stage) followed by a lightweight linear attention module.*



(a) Convolutional modules (b) Transposed convolution (c) Attention mechanisms

Fig. 3: ***Module Throughput**: Throughput evaluation of convolutional, transposed convolutional, depthwise convolutional, and attention-based modules across varying input resolutions on an NVIDIA 4070 [25] with FP32 precision.*

that are intrinsically well-suited to parallel execution. For example, both map and reduction operations have a step complexity of $O(N)$ on a CPU, whereas on a GPU, map executes in $O(1)$ and reduction in $O(\log N)$, where N denotes the input size [1].

The penalty term is similar to the works of [11] and is incorporated into the training objective as a hinge-style regularizer, resulting in the overall loss function, where L_0 is the original loss of the model and γ acts as the compression factor over the module size normalizing constant C:

$$\text{Loss} = L_0 + \frac{\gamma}{C} \sum_{k=1}^{\text{all modules}} \alpha_k \text{size}_k \tag{1}$$

For our experiments, we use quantization approach (from [8]) to quantize weights in the forward pass and use symmetric differentiable number Q8A format [23] as shown in Equation 2. The bit depths (b) and the scaling factor (e) are set to be trainable and is broadcasted to the weights of the tensor (W) during the forward pass of the training.

$$Q(W, b, e)_k = 2^e \left\lfloor \text{clip}\left(\frac{W}{2^2}, -2^{b-1}, 2^{b-1} - 1\right)\right\rceil \tag{2}$$

The $\lfloor \cdot \rceil$ acts as a straight-through estimator [4] for the rounding function, which returns the identity of upstream gradient during the backward pass.

We now present our proposed method of derivation for the penalty cost of the compute modules in our transformer block.

Convolution serve as the foundational operation within our transformer blocks, providing a strong local inductive bias essential for vision tasks. Although full softmax attention [34] excels at capturing both local and long-range dependencies, its computational requirements are prohibitive for high-resolution images. The standard attention mechanism scales as $O((HW)^2 dp^{-2})$ for an image of spatial dimensions $H \times W$, embedding dimension d, and patch size p, resulting in quadratic complexity with respect to spatial resolution.

In contrast, the convolutional layers in our architecture offer a significantly more efficient alternative. A convolutional kernel of shape (O, I, k, k) performs $O(I \cdot H \cdot W \cdot k^2)$ operations (*quadratic only on kernel size*). Crucially, it exhibits constant time step complexity (parallel time) due to its highly parallel nature. This efficiency stems from the small kernel sizes, which allow the operation to be executed via a fast, within-block reduction on the accelerator. On resource-constrained devices or when dealing with high resolutions images, compute scheduling is needed, which scales linearly with spatial resolution and input/output modes.

We enhance this efficiency through learned quantization. Each output channel mode of a tensor O_i is associated with a learnable bit-depth b_i and exponent e_i. The quantized weights are returned by applying the quantization function (Equation 2) to the full-precision kernel.

$$\text{size}(l) = IHW \sum_{i=1}^{O} \max(0, \lceil b_{i,l} \rceil) \tag{3}$$

Equation 3 shows that penalizing the parallel step complexity of convolution is equivalent to penalizing its tensor size (storage cost). As a result, the penalty discourages allocating high precision across many output channels and instead promotes sparse filters within the filter bank.

Depthwise Convolution One of the ways to improve throughput in convolutional layers is to apply strided convolution, which reduces the work linearly with spatial resolution, though at the expense of reducing the effective local receptive field. Another option is to use grouped convolution, where the input channels are divided into g groups. In this case, the convolutional kernel has the shape $(O, I/g, k^2)$, with g denoting the number of groups. A special case of this is depthwise convolution, where the number of groups g equals the number of input modes I. As shown in Figure 3a, depthwise convolution [40] has higher throughput than ungrouped convolution, specially when spatial resolution increases.

In this work, we employ depthwise convolutions in the expansion phase of the vision tower (Figure 2). This ensures that the computational cost does not scale with the embedding dimension, unlike ungrouped convolutions. To regularize these layers, we apply the same heuristic penalty formulation as in the ungrouped convolution mentioned above.

Transposed Convolution For the upscaling stage, we employ transposed convolution after the projection block to generate upscaled features from expanded convolution outputs [32]. Although transposed convolutions are more expensive than standard convolutions (Figure 3b), we opt for ungrouped transposed operations since they provide richer local features than depthwise alternatives. To mitigate cost, we restrict the number of output channels rather than performing full ungrouped upscaling. This strategy follows the bottleneck principle in ResNet [15] and the reduced-channel decoding of U-Net [29], where only a limited set of upscaled features is sufficient to recover fine spatial details. In the subsequent stages, we apply one final composition of depthwise convolution and strided convolution to efficiently downscale the features for the linear attention module.

Linear Attention While softmax attention captures both local and global dependencies, we expexct that local context is already captured by the composition of convolution functions in the previous stages of our transformer block. The linear attention module therefore focuses exclusively on global receptive fields. Unlike standard attention, which scales quadratically with spatial resolution, linear attention computes global interactions in time linear with respect to spatial resolution [37, 19].

$$Q \in \mathbb{R}^{B \times N \times d}, \quad K \in \mathbb{R}^{B \times N \times d}, \quad V \in \mathbb{R}^{B \times N \times d}, \tag{4}$$

$$\mathrm{Att}(Q,K,V) \;=\; \frac{\phi(Q)\big(\phi(K)^\top V\big)}{\phi(Q)\big(\phi(K)^\top \mathbf{1}_n\big)}, \quad \phi(x)=\mathrm{ELU}(x)+1 \tag{5}$$

In the context of parallel accelerators, the computational cost of linear attention reduces to a sequence of matrix multiplications, since the attention scores are obtained using a constant-time activation function (like ELU(x)). A matrix multiplication can be decomposed into parallel dot-products, each of which corresponds to a reduction operation. On a parallel machine with R inputs, such reductions can be performed hierarchically in $O(\log R)$ steps [13]. As a result, the effective depth of a matrix multiplication scales only logarithmically with the dimension of the reduction. For instance, computing $K^\top V$ has a depth complexity of $O(\log N)$, where N denotes the spatial $(H \cdot W)$, while multiplying Q with the result incurs a depth complexity of $O(\log d)$.

We leverage per-head quantization of queries, keys, values, and outputs, which can be broadcasted prior to computation. This design enables integration with off-the-shelf implementation of Flash Linear Attention [3], improving runtime throughput while preserving linear complexity.

$$\mathrm{size}_{\mathrm{attn}_l} = log(d \cdot H \cdot W) \cdot \sum_{i=1}^{heads} \max(0, \lceil b_{head_i,l} \rceil) \tag{6}$$

In Equation 6, $b_{head_i,l}$ is the learned bit-depth for the i-th output channel (head) of the query weight matrix. This formulation allows the model to learn mixed-precision strategies per attention head, potentially allocating higher precision to more sensitive components of the attention mechanism. (*Note:* $\log(d \cdot HW) \sim \log(HW)$, as the order of spatial resolution HW is much higher than the embedding dimension d.)
For the final MLP heads that project features to the final output logits, we apply quantization to the weight matrices in order to enable low-precision inference. However, no additional compression penalty is imposed on these layers ($size_{mlp} = 0$) since they do not significantly contribute to the overall latency of the model.

4 Experiments

4.1 Experimental Setup

Target Device and Throughput Measurement A core contribution of our work is the integration of hardware-aware compression into the training objective. We profile the throughput of each computational module on our target device, an NVIDIA RTX 4070 Mobile GPU [25], at FP8 precision with a standardized batch size to establish a baseline.

While profiling is on the mobile GPU, training is performed on a single NVIDIA H100 GPU [24]. For Compute-bound operations, such as linear attention, we modify fused kernels [38] to support quantization-aware training, whereas for memory-bound operations like top-k selection and cross-entropy, we rely on off-the-shelf optimized libraries [10] to improve training time.

Model Implementation and Training Protocol Our architecture is implemented in PyTorch [27], leveraging fused kernels via CUTLASS [7] and Triton [18]. All models are optimized with AdamW [21] ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, weight decay 0.01). We use a cosine annealing schedule for the learning rate, decaying from 10^{-3} to 10^{-5}, while the compression coefficient γ ramps from 0.1 to 0.2 on a separate cosine schedule, ensuring stable convergence before introducing significant compression pressure.

All models are trained from scratch with parameters initialized to simulate 4-bit precision for stable quantization-aware training. Training runs for up to 1000 epochs with early stopping (patience 50, minimum delta 0.01 on validation loss).

4.2 Datasets

We evaluate our method across three standard dense prediction tasks: *image classification*, *semantic segmentation*, and *optical flow*, chosen to test both local feature extraction and global contextual reasoning (Table 1). All datasets are publicly available. We follow the official train-validation-test splits provided by the original dataset. Both rectangular and square images are normalized and resized to 448×448 pixels using bilinear interpolation, and augmented with random cropping to improve robustness and prevent overfitting, ensuring consistent evaluation across tasks.

Classification We use two benchmarks for classification: CUB-200-2011 [35] for fine-grained species recognition (11,788 images of 200 bird classes) and Country211 [28] for geo-location classification (63,000 images across 211 countries). CUB emphasizes subtle local visual differences, while Country211 evaluates global context recognition and long-term memory capacity (Figure 4).

Segmentation For semantic segmentation, we adopt ADE20K [41], containing 20,000 training images and 2,000 validation images annotated across 150 semantic categories (Figure 5).

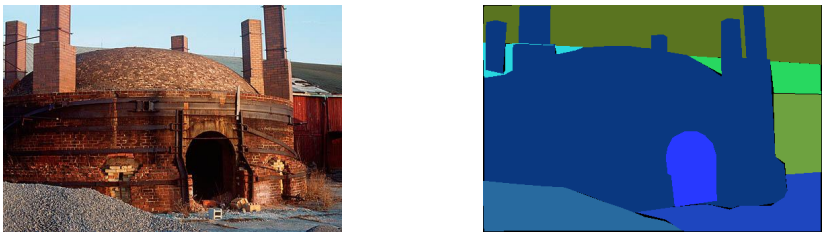
Optical Flow For optical flow estimation, we use HD1K [20], which provides 1,066 high-definition frames with dense ground truth flow fields. Frames are resized to 448×448 pixels for consistency across experiments.

Table 1: Summary of datasets used in our experiments.

Dataset	Task	#Images	Resolution	Classes / Channels
CUB-200-2011	Classification	11,788	500×500	200 species
Country211	Classification	63,000	Variable	211 countries
ADE20K	Segmentation	22,000	Variable	150 categories
HD1K	Optical Flow	1,066 frames	1920×1080	Flow fields

(a) *Hooded Warbler* (b) *Golden-winged* (c) *Urban landscape* (d) *Natural scenery*

Fig. 4: *Representative examples from the classification datasets: CUB-200-2011 (birds a), b)) and Country211 (landscapes from the same country label 0 c), d)).*



(a) Input image (b) Semantic mask

Fig. 5: *Example from ADE20K dataset showing input image and the output mask.*

4.3 Ablation Studies

We perform ablation studies on a compact 1.98M-parameter model evaluated on CUB-200-2011 [35] to isolate the effects of individual architectural components on performance and compression.

Pixel Shuffling vs. Transposed Convolution We evaluate pixel shuffling [31] as a non-learnable alternative to transposed convolution for feature upscaling. While pixel shuffling rearranges the input tensor to generate higher resolution features without additional compute costs, it inherently couples feature channels, preventing independent pruning or precision reduction. This coupling further limits the use of non-uniform compression in the preceding projection convolutions, as it would otherwise yield uniformly noisy upscaled outputs. For this reason, we retain transposed convolution for upscaling, since it preserves channel-wise independence that is essential for effective, precision-guided pruning.

Pointwise vs. 3×3 Convolution for Projection Pointwise (1×1) convolutions are parameter-efficient, but quantized low-bit kernels tend to collapse feature representations early in training, disrupting gradient flow. Using 3×3 kernels mitigates this issue by providing additional spatial context while maintaining efficiency. In our experiments, quantized 3×3 kernels achieve stable training

without degradation in final performance when compared to an unquantized 1x1 pointwise kernels.

5 Evaluation

To evaluate the proposed method, we assessed the task performance and computational efficiency. When assessing task performance, we report **Top-1** and **Top-5** accuracy, capturing exact and lenient recognition capabilities for classification task [35]. For semantic segmentation, **mean Intersection over Union (mIoU)** [12] evaluates the overlap between predicted and ground truth segments, averaged across all classes. For optical flow, **Average End-Point Error (EPE)** [2] measures the mean Euclidean distance between predicted and true flow vectors, capturing both magnitude and directional errors. For all metrics, we record evaluations at full precision (FP32) and in automatic mixed precision (FP8) setting to assess the impact of reduced precision. For computational efficiency, we report **FLOPS** per forward pass as a theoretical complexity measure [33], and **throughput** (images/sec at FP8) on NVIDIA RTX 4070 Mobile, capturing real-world deployment performance.

5.1 Performance on Classification Tasks

Results on CUB-200-2011 (Table 2) and Country211 (Table 3) show that our approach maintains strong accuracy while improving efficiency in both FP32 and FP8. On CUB-200-2011, our models frequently outperform EfficientViT and DiffQ in Top-1/Top-5 accuracy with fewer parameters. The 0.88M model with FP8 inference achieves the highest throughput across all models, while larger variants match or exceed EfficientViT accuracy with higher efficiency. Under FP8, our accuracy remains stable ($< 1\%$ drop on average across all sizes). In contrast, EfficientViT suffers sharp FP8 degradation (up to 6%), and DiffQ lags in accuracy and only modestly benefits from FP8.

On Country211, accuracy differences are smaller, but our models still achieve better Top-1/Top-5 accuracy than DiffQ and higher throughput than EfficientViT. FP8 again confirms robustness: our accuracy remains steady while throughput increases, whereas EfficientViT loses up to 8–9% Top-1 accuracy under FP8.

5.2 Performance on Segmentation

On ADE20K (Table 4), our method achieves strong compression while preserving accuracy. With 4.28M parameters, it matches EfficientViT in mIoU (43.7 vs. 44.1) while reducing size by $\sim$9% and FLOPs by 6%, and delivers the highest throughput (282 img/s). FP8 results show sharper contrasts: EfficientViT drops drastically (44.1 $\rightarrow$ 32 mIoU), while DiffQ is more stable but less accurate overall. Our method remains robust, with only a minor decline (43.7 $\rightarrow$ 43.3) and the best throughput.

Table 2: Comparison of our models on the CUB-200-2011 dataset [35]. Throughput is measured in FP8.

Model	**Size**	**Top-1 (%) ↑**		**Top-5 (%) ↑**		**FLOPs (G) ↓**	**Throughput (img/s) ↑**
		FP32	FP8	FP32	FP8		FP8
EfficientViT	0.98M	71.3	66.1	95.5	90.2	7	309
DiffQ	1.12M	68.1	68.0	92.3	92.5	9	281
Ours	**0.88M**	**73.4**	**73.4**	**98.0**	**97.2**	**7**	**338**
EfficientViT	2.41M	76.2	72.0	100	96.5	24	262
DiffQ	2.50M	71.3	70.0	100	94.1	28	276
Ours	**1.98M**	**75.8**	**74.9**	**100**	**100**	**24**	**308**
EfficientViT	4.81M	81.5	75.5	100	98.0	48	183
DiffQ	4.92M	78.1	76.4	100	96.2	54	172
Ours	**4.68M**	**83.5**	**83.4**	**100**	**99.3**	**51**	**230**

Table 3: Benchmark results on the Country211 dataset [28]. Throughput is measured in FP8.

Model	**Size**	**Top-1 (%) ↑**		**Top-5 (%) ↑**		**FLOPs (G) ↓**	**Throughput (img/s) ↑**
		FP32	FP8	FP32	FP8		FP8
EfficientViT	2.41M	44.4	41.2	65.6	52.1	23	291
DiffQ	2.50M	36.0	35.8	41.1	40.4	28	270
Ours	**2.13M**	**43.5**	43.5	**65.3**	63.1	**25**	**324**
EfficientViT	4.81M	52.1	44.9	80.2	69.5	48	228
DiffQ	4.92M	38.4	38.3	65.2	65.5	54	213
Ours	**4.68M**	**55.7**	**54.9**	**81.0**	**81.3**	**51**	**294**

Table 4: *Semantic segmentation results on the ADE20K dataset [41], where mIoU is mean Intersection over Union. Throughput is measured in FP8.*

Model	**Size**	**mIoU (%) ↑**		**FLOPs (G) ↓**	**Throughput (img/s) ↑**
		FP32	FP8		FP8
EfficientViT	4.68M	44.1	32.0	47	255
DiffQ	4.92M	40.1	39.5	54	241
Ours	**4.28M**	**43.7**	**43.3**	**44**	**282**

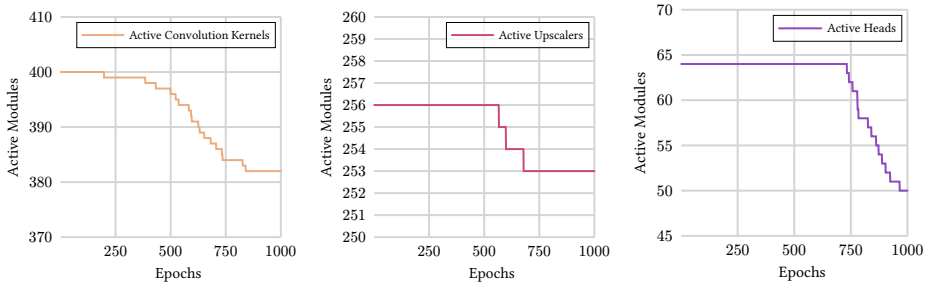
5.3 Performance on Optical Flow

On HD1K (Table 5), EfficientViT achieves the lowest FP32 EPE (4.8), but our approach is competitive (5.0) with lower FLOPs, and much higher throughput (227 img/s vs. 191). DiffQ performs worse in both accuracy (5.4) and speed.

In FP8, EfficientViT degrades sharply (4.8 → 7.2 EPE), showing poor quantization robustness. DiffQ is more stable but less accurate. Our method shows only a modest increase (5.0 → 5.2) while retaining the best throughput, demonstrating that it compresses effectively and adapts better to low-precision inference, making it well-suited for real-time flow estimation.

Table 5: *Optical flow estimation results on the HD1K dataset [20], where EPE is End Point Error. Throughput is measured in FP8.*

Model	**Size**	**EPE ↓**		**FLOPs (G) ↓**	**Throughput (img/s) ↑**
		FP32	FP8		FP8
EfficientViT	4.81M	4.8	7.2	48	191
DiffQ	4.92M	5.4	5.8	54	155
Ours	**4.68M**	**5.0**	5.2	**51**	**227**

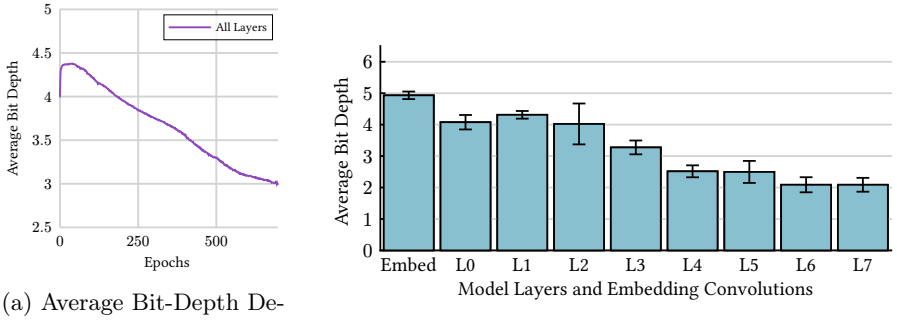


(a) Pruning of convolutional weights (b) Pruning of upscaling layers (c) Pruning of attention heads

Fig. 6: *Pruning effectiveness across architectural components. Convolutional filters and attention heads undergo significant reductions, while upscalers are minimally pruned.*

5.4 Quantization and Compression Trends

In this section, we analyze pruning and quantization in our 1.98M model trained on CUB-200-2011. Figure 6 shows distinct pruning patterns. Convolutional kernels shrink gradually after ∼300 epochs, accelerating between 500–750, with active kernels dropping from 400 to ∼382 (≈ 4.5%). Transposed convolution upscalers are barely affected, with only three drops between 500–700 epochs (256 → 253, ≈ 1.2%), reflecting their importance as upscaled features. Attention heads face



(a) Average Bit-Depth Decay Across Epochs

(b) Learned Bit-Depth Allocation Across Layers

Fig. 7: *Quantization-aware training dynamics. (a) Progressive reduction of bit-depth variance throughout training. (b) Heterogeneous precision allocation (higher precision in sensitive early layers and aggressive compression in deeper layers.)*

the strongest compression, staying near 64 until 750 epochs before plunging to ∼50 (≈ 21.9%). Overall, pruning favors preserving resolution-critical modules while heavily reducing attention.

Quantization (Fig. 7) shows bit-depth decays smoothly from ∼4.3 to ∼3 bits, with an early plateau. The final allocation is heterogeneous: embedding convolution kernel gain bit depth from initialization (4 →5 bits) suggesting degraded perfomance if inference performed on an even lower precision setting such as FP4, early/mid layers (L0–L2) stay near 4, L3–L5 drop to 2.5–3, and deeper layers (L6–L7) operate at ∼2 bits. Early features thus remain precision-sensitive, while deeper layers show tolerance to aggressive compression.

6 Discussion

Our results strongly support the effectiveness of cost-aware, self-compressing vision architectures. By selectively pruning and applying quantization based on the computational cost of each module, our model matches the accuracy of EfficientViT while consistently improving throughput. Compression is applied non-uniformly, with more aggressive pruning and lower bit-depths assigned to less critical modules, validating the hypothesis that module-specific computational cost can guide compression strategies. Across fine-grained classification, semantic segmentation, and optical flow tasks, the compressed model maintains performance comparable to non-compressed baselines, indicating that contemporary architectures contain substantial redundancy that can be eliminated without sacrificing model perfomance.

In addition to reducing storage through pruning and learned quantization, our approach improves throughput compared to DiffQ, with full speedup realized after graph re-compilation. The compression patterns also reveal structural insights: later layers converge to lower bit-depths, suggesting their reduced contribution to overall performance, while structured pruning assigns varying precision within

the same layer, reflecting heterogeneous importance across compute modules. These trends not only guide the design of efficient, hardware-aware architectures but also indicate safe lower-precision regimes for deployment.

7 Limitations and Future Work

Despite the promising results, several limitations remain that point to directions for future research. First, improvements in inference throughput are only fully realized post-training through graph compilation. Since the current loop does not recompile after modules are pruned, the training-time graph fails to reflect evolving sparsity, leaving potential latency gains unexploited during training. Second, the analysis was restricted by computational constraints to a single model scale for the more demanding tasks of optical flow estimation and semantic segmentation. Extending the study to multiple scales (Small, Base, Large) would clarify how well the method generalizes across architectures of varying complexity. Last, while the proposed approach improves throughput and storage, real latency benefits from mixed-precision computation are hardware-dependent, often requiring dedicated support for low-bit operations. Future work can examine the generalizability of the compression stratergy with specific deployment hardware to realize improvement in throughput.

8 Conclusion

This work proposed a novel, self-compressing vision tower that strategically prunes and quantizes all major modules convolutions, transposed convolutions, and linear attention based on their computational cost and contribution to the network's output. The model's performance on classification, segmentation, and optical flow benchmarks consistently matches the accuracy of the state-of-the-art EfficientViT, while often providing the added benefits of reduced storage and improved throughput. Furthermore, the proposed method consistently surpasses the baseline quantization method, DiffQ, in both accuracy and throughput.

Beyond its practical utility, we conclude that this constructive, cost-aware approach to compression serves a dual purpose: it efficiently reduces model size and computational demand, and it also provides a novel methodological tool for inspecting module importance across layers. The insights gained from the heterogeneous compression patterns can directly inform and facilitate more effective and efficient model design for both training and inference on specific target devices.

Bibliography

[1] Attention is logarithmic, actually (2025), https://supaiku.com/attention-is-logarithmic

[2] Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M.J., Szeliski, R.: A database and evaluation methodology for optical flow. In: International Conference on Computer Vision (ICCV). pp. 233–236 (2011). https://doi.org/10.1109/ICCV.2011.6126252

[3] Beck, M., Pöppel, K., Lippe, P., Hochreiter, S.: Tiled flash linear attention: More efficient linear rnn and xlstm kernels (2025), https://arxiv.org/abs/2503.14376

[4] Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation (2013), https://arxiv.org/abs/1308.3432

[5] Cai, H., Li, J., Hu, M., Gan, C., Han, S.: Efficientvit: Multi-scale linear attention for high-resolution dense prediction (2024), https://arxiv.org/abs/2205.14756

[6] Canziani, A., Paszke, A., Culurciello, E.: An analysis of deep neural network models for practical applications (2016), arXiv preprint arXiv:1605.07678

[7] Corporation, N.: Cute: Cuda tensor expressions. https://docs.nvidia.com/cutlass/media/docs/cpp/cute/index.html (2025), accessed: 2025-08-18

[8] Cséfalvay, S., Imber, J.: Self-compressing neural networks (2023), https://arxiv.org/abs/2301.13142

[9] Dao, T.: Hybrid linear-softmax attention working very well at large scale and long-context! as we've seen with multiple models now .... https://x.com/tri_dao/status/1879439184462762225 (Jul 2025), https://x.com/tri_dao/status/1879439184462762225, tweet

[10] Dao-AILab: QuACK: A quirky assortment of cute kernels. https://github.com/Dao-AILab/quack (2025), open-source CUDA kernel collection written in Python using CuTe-DSL. Apache-2.0 license. Accessed: 2025-08-18

[11] Défossez, A., Adi, Y., Synnaeve, G.: Differentiable model compression via pseudo quantization noise (2022), https://arxiv.org/abs/2104.09987

[12] Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International Journal of Computer Vision **88**(2), 303–338 (2010). https://doi.org/10.1007/s11263-009-0275-4

[13] Goto, K., van de Geijn, R.A.: Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software (TOMS) **34**(3), 12:1–12:25 (2008). https://doi.org/10.1145/1356052.1356053

[14] Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., Roberts, D.A.: The unreasonable ineffectiveness of the deeper layers (2025), https://arxiv.org/abs/2403.17887

[15] 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 (CVPR). pp. 770–778 (2016)

[16] Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)

[17] 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)

[18] Jiang, J., Zhu, J., Chong, M., Agarwal, S., Hsiao, C.H., Wang, Y., Diao, C., Wu, T., Singh, A.: Triton: An intermediate language and compiler for optimizing deep learning on gpus. In: Advances in Neural Information Processing Systems (NeurIPS) 35 (2022), https://openreview.net/forum?id=7phvJ2kvEY

[19] Katharopoulos, A., Vyas, A., Pappas, N., Fleuret, F.: Transformers are rnns: Fast autoregressive transformers with linear attention (2020), https://arxiv.org/abs/2006.16236

[20] Kondermann, D., Nair, R., Honauer, K., Kuhn, A., Schöps, T., Schmidt, J., Nowozin, S., Geiger, A., Rother, C., Kondermann, C.: The hci benchmark suite: Stereo and flow ground truth with uncertainties for urban autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). pp. 19–28 (2016). https://doi.org/10.1109/CVPRW.2016.121

[21] Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: International Conference on Learning Representations (ICLR) (2019), https://openreview.net/forum?id=Bkg6RiCqY7

[22] Mehta, S., Rastegari, M.: Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer (2022), https://arxiv.org/abs/2110.02178

[23] Micikevicius, P., Stosic, D., Burgess, N., Cornea, M., Dubey, P., Grisenthwaite, R., Ha, S., Heinecke, A., Judd, P., Kamalu, J., Mellempudi, N., Oberman, S., Shoeybi, M., Siu, M., Wu, H.: Fp8 formats for deep learning (2022), https://arxiv.org/abs/2209.05433

[24] NVIDIA Corporation: Nvidia h100 tensor core gpu. https://www.nvidia.com/en-us/data-center/h100/ (2022), accessed: 2025-08-18

[25] NVIDIA Corporation: Nvidia geforce rtx 4070 laptop gpu. https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4070-laptop/ (2023), accessed: 2025-08-18

[26] Palmas, A., Andronico, P.: Deep learning computer vision algorithms for real-time uavs on-board camera image processing. arXiv preprint arXiv:2211.01037 (2022)

[27] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep

learning library. In: Advances in Neural Information Processing Systems 32 (NeurIPS 2019) (2019)
[28] Radford, A., Kim, J.W., Xu, T., Brockman, G., McLeavey, C., Sutskever, I.: Country211 dataset. https://openaipublic.azureedge.net/clip/data/country211.tgz (2021), subset of YFCC100M filtered by GPS coordinates mapped to ISO-3166 country codes. Released as part of the CLIP dataset resources. Use subject to underlying Creative Commons licenses.
[29] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention (MICCAI). pp. 234–241. Springer (2015)
[30] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks (2019), https://arxiv.org/abs/1801.04381
[31] Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A.P., Bishop, R., Rueckert, D., Wang, Z.: Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network (2016), https://arxiv.org/abs/1609.05158
[32] Shocher, A., Feinstein, B., Haim, N., Irani, M.: From discrete to continuous convolution layers (2020), https://arxiv.org/abs/2006.11120
[33] Sovrasov, V.: ptflops: a flops counting tool for neural networks in pytorch framework (2018-2024), https://github.com/sovrasov/flops-counter.pytorch
[34] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2023), https://arxiv.org/abs/1706.03762
[35] Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: Cub-200-2011 (Apr 2022). https://doi.org/10.22002/D1.20098
[36] Wu, K., Zhang, J., Peng, H., Liu, M., Xiao, B., Fu, J., Yuan, L.: Tinyvit: Fast pretraining distillation for small vision transformers. In: European conference on computer vision (ECCV) (2022)
[37] Xu, Y., Li, C., Li, D., Sheng, X., Jiang, F., Tian, L., Barsoum, E.: Qt-vit: improving linear attention in vit with quadratic taylor expansion. In: Proceedings of the 38th International Conference on Neural Information Processing Systems. NIPS '24, Curran Associates Inc., Red Hook, NY, USA (2025)
[38] Yang, S., Zhang, Y.: Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism (Jan 2024), https://github.com/fla-org/flash-linear-attention
[39] Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks (2013), https://arxiv.org/abs/1311.2901
[40] Zhang, W., Li, J., Wang, Y.: Real-time image processing in iot devices with a lightweight yolo approach. In: Proceedings of the International Conference on Computer Vision and Pattern Recognition. pp. 1234–1242 (2025)
[41] Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., Torralba, A.: Scene parsing through ade20k dataset. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5122–5130 (2017). https://doi.org/10.1109/CVPR.2017.544