EVAR: Edge Visual Autoregressive Models via Principled Pruning

Zefang Wang[1,2], Yanyu Li[3], Mingluo Su[1], Simin Xu[1], Guanzhong Tian[2,†], Huan Wang[1,†]
[1]Westlake University [2]Zhejiang University [3]Snap Inc.

Abstract

*Recent advances in generative modeling have catalyzed demand for on-device single-image synthesis. However, the stringent compute and memory budgets of resource-constrained edge hardware hinder the deployment of large-scale models. Next-scale visual autoregressive (VAR) models—which predict finer-scale content conditioned on coarser resolutions—offer strong fidelity, generalization, and improved inference efficiency, yet remain costly to run on such devices. We introduce EVAR, an efficient structured-pruning framework tailored to next-scale VAR models and edge deployment. EVAR instantiates a principled pruning paradigm: it couples Optimal Brain Surgeon–guided, Hessian-aware sensitivity estimation with closed-form weight updates, and augments them with scale-aligned calibration and compensation. By grounding pruning decisions in second-order optimality and executing updates analytically, EVAR mitigates compression-induced degradation while preserving next-scale conditioning—turning sparsification from a heuristic into a disciplined procedure. To further address scale-wise gradient and loss imbalance during fine-tuning, we propose Progressive Scale-Aware Distillation (PSAD), which leverages VAR's multi-scale generative hierarchy to reweight scales and enforce cross-scale consistency in the pruned model. On ImageNet single-image generation benchmarks, EVAR reduces parameter count, memory footprint, and end-to-end latency while retaining competitive generative quality. On an iOS deployment, EVAR further cuts single-image latency from 494 ms to 277 ms **(1.8× speedup)**, with FID changing only marginally.*

1. Introduction

Autoregressive (AR) [7, 21, 42, 51] models have recently made remarkable progress in visual generation, where their strong scalability and generalization enable high-quality image synthesis and editing. However, conventional AR models rely on a next-token prediction paradigm and generate tokens strictly sequentially, so the token-by-token decoding requires numerous steps and leads to significant inference latency.

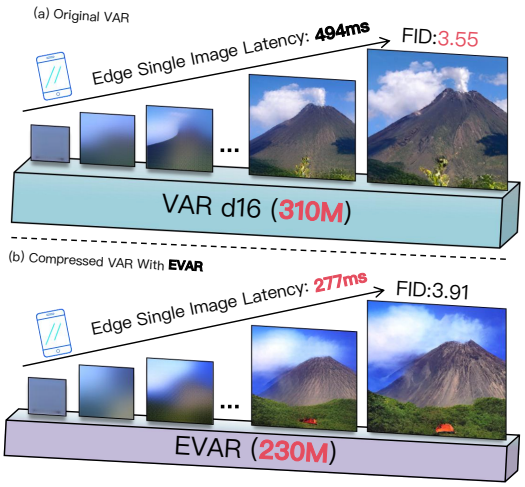


Figure 1. EVAR compresses a next-scale VAR-d16 backbone for edge deployment. (a) Original VAR-d16 with 310M parameters achieves single-image latency of 494 ms with FID 3.55. (b) After EVAR's structured pruning, the model is reduced to 230M parameters and reaches 277 ms latency on iPad Pro (M4), corresponding to about 1.8× speedup with only ∼10% relative FID degradation.

Visual autoregressive (VAR) modeling [5, 6, 13, 23, 24, 27, 37, 43, 44, 49, 52–54] shifts this paradigm from next-token to next-scale prediction. Instead of emitting tokens one by one, VAR decodes visual content in a coarse-to-fine multi-scale hierarchy, generating many tokens in parallel within each scale. This design drastically reduces the number of decoding steps while retaining the expressive power of AR models, enabling performance comparable to state-of-the-art diffusion models on image synthesis, super-resolution, and inpainting. Yet these models remain large and computationally demanding, making on-device deploy-

† Corresponding authors

ment on resource-constrained edge hardware—where per-image latency and memory are critical—particularly challenging. In this work, we focus on single-image generation in such edge settings.

These constraints motivate efficient model compression techniques [4, 8, 15, 17, 47, 48, 50] that can reduce the computational and memory footprint without significantly degrading generative quality. However, existing pruning methods are mainly developed for classification networks or large language models and do not account for the multi-scale structure and generation mechanism of VAR.

To address this gap, we propose *EVAR*, a structured compression framework tailored to VAR and edge deployment. EVAR adopts an OBS-guided [14] adaptive pruning-and-compensation scheme that assigns block-wise pruning ratios based on Hessian information and explicitly compensates for the effect of removed weights. By coupling pruning decisions with each block's internal structure and sensitivity, EVAR substantially reduces parameters and computation while preserving generation quality, making VAR more suitable for deployment on edge devices.

Beyond pruning, we find that next-scale VAR suffers from inherent scale-wise gradient imbalance: high-resolution scales contain far more tokens than coarse scales, so standard training objectives over-emphasize fine details while under-supervising coarse semantics, an issue further exacerbated after pruning. EVAR addresses this with a *progressive scale-aware distillation* (PSAD) scheme that combines a coarse-to-fine distillation curriculum with scale-aware loss weighting, rebalancing gradients across scales, and improving the fine-tuning of pruned VAR models.

To evaluate EVAR under realistic deployment conditions, we build an end-to-end on-device pipeline for single-image VAR generation on iOS. We convert pruned models using Apple's official toolchain, run them in a Swift-based iOS application with carefully selected compute units, and measure end-to-end latency and memory usage. On ImageNet, EVAR delivers competitive generative performance while enabling efficient mobile and edge deployment of VAR models.

Our contributions can be summarized as follows:

- We introduce *EVAR*, the first OBS-guided adaptive pruning and compensation framework tailored for VAR models, providing a principled structured pruning pipeline for edge deployment.
- We propose progressive scale-aware distillation (PSAD), which addresses scale-wise gradient imbalance in next-scale VAR and improves the fine-tuning of pruned models via a coarse-to-fine, scale-weighted distillation.
- Empirically, we build a practical on-device deployment and evaluation pipeline for single-image VAR generation on iOS with CoreML inference engine, reporting 1.8× speedup with comparable generation quality.

2. Related Work

2.1. Efficient Auto-regressive Image Generation

Autoregressive (AR) models decompose the joint distribution of an image into a product of conditional distributions and have been a long-standing backbone of generative modeling. Early work, such as PixelRNN and PixelCNN [31, 45] generates pixels sequentially with recurrent or convolutional architectures, but suffers from slow sampling. With the advent of Transformer-based language models [2, 32, 33, 35, 36, 46], large-scale GPT-style decoders have been adopted for images, enabling global context modeling and strong scalability. Recent systems, including VQGAN and RQ-Transformer [7, 19], diffusion–AR hybrids [11, 55], and GPT-like image generators such as LlamaGen and Lumina-mGPT [26, 42] further push fidelity by operating in discrete token spaces with powerful next-token prediction.

To reduce the sequential bottleneck, a line of work studies partially parallel or masked decoding, e.g., MaskGIT and MAR [3, 22], and scalable diffusion backbones such as DiT and LDM [34, 38]. CoDe [6] boosts efficiency with minimal quality loss via collaboration: large drafter for small-scale low-frequency, small refiner for large-scale high-frequency details. This design achieves competitive quality with substantially fewer decoding steps, and has been extended to strong AR baselines such as PAR and AiM [21, 51]. However, most of these works target fidelity and scalability on powerful GPUs; compression and practical deployment of next-scale VAR on resource-constrained edge devices remain largely unexplored, which is the focus of EVAR.

2.2. Network Pruning

Network pruning [12, 30] is a standard way to reduce computation and memory by removing redundant parameters from a pre-trained network. According to granularity, methods are typically divided into *unstructured* and *structured* pruning. Unstructured pruning [9, 10, 14, 18, 39, 40] zeros out individual weights according to saliency scores, enabling fine-grained parameter removal at the level of each matrix entry. This granularity often yields lower accuracy degradation at a given sparsity compared to structured pruning. However, the resulting irregular sparsity is difficult to exploit on general-purpose hardware due to poor alignment with SIMD/SIMT execution, irregular memory access patterns, and limited kernel support; consequently, end-to-end speedups are frequently modest despite high nominal sparsity. Structured pruning [1, 8, 16, 20, 25, 28, 41], removes entire channels or heads, yielding dense, smaller subnetworks that better align with standard conv/GEMM kernels and thus more directly translate into wall-clock acceleration.

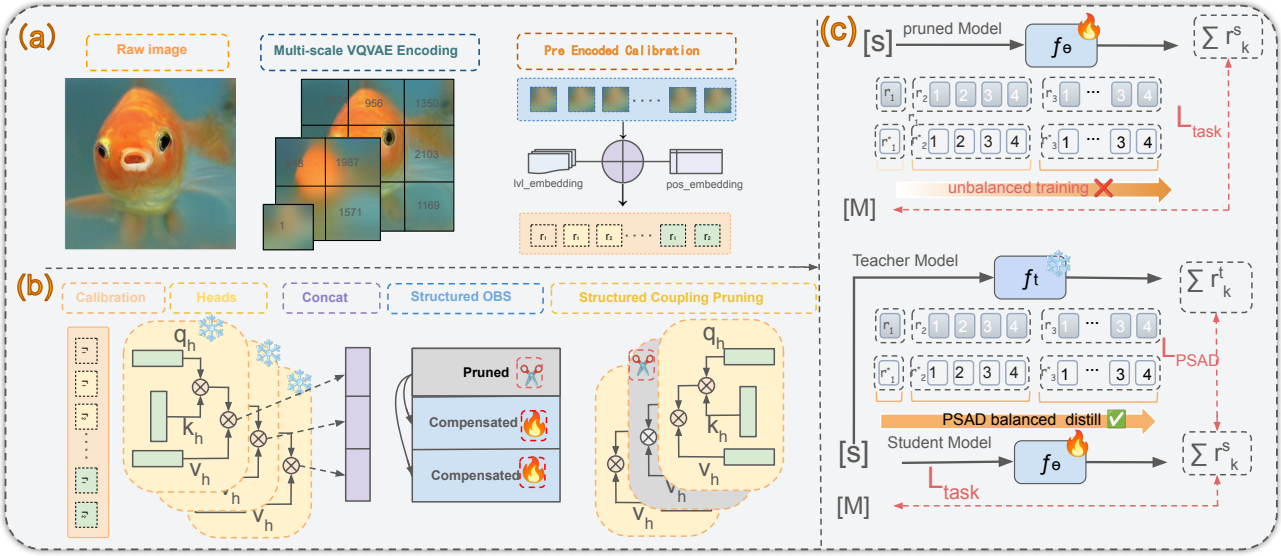


Figure 2. (a) *Pre-encode calibration:* a raw image is encoded by the VQ-VAE into a multi-scale residual pyramid, which is then converted into next-scale VAR tokens with level and positional embeddings to form the pre-encoded calibration set. (b) *OBS-guided structured pruning:* using the input covariance of the output projection O to approximate its Hessian, EVAR applies structured OBS to prune attention heads, compensates the remaining heads in closed form, and finally removes the corresponding coupled heads in the shared Q/K/V projections. (c) *Progressive Scale-Aware Distillation (PSAD):* standard fine-tuning or vanilla distillation suffers from scale-wise loss imbalance in next-scale VAR, whereas PSAD reweights and schedules the multi-scale distillation loss to balance supervision across scales, yielding state-of-the-art post-pruning recovery.

3. Method

3.1. Preliminaries on VAR

Visual Autoregressive (VAR) models [44] reformulate autoregressive image modeling by shifting from traditional next-token prediction to a next-scale prediction scheme. Instead of generating an image token by token, VAR divides the input feature map $f \in \mathbb{R}^{h\times w\times C}$ into K token maps $(r_1, r_2, \ldots, r_K)$ at multiple resolutions, where the spatial resolution increases with the scale index and the final token map r_K recovers the original feature-map resolution.

The joint distribution over the multi-scale token maps is factorized as

$$p(r_1, r_2, \ldots, r_K) = \prod_{k=1}^{K} p(r_k \mid r_1, \ldots, r_{k-1}), \qquad (1)$$

where $r_k \in [V]^{h_k\times w_k}$ denotes the token map at scale k, and $(r_1, \ldots, r_{k-1})$ provides the coarse-to-fine context for predicting r_k. At each autoregressive step k, all tokens in r_k are generated in parallel, conditioned on the previous scales and their positional embeddings.

3.2. OBS-Guided Structured Pruning for VAR

Network pruning reduces model size and inference cost by removing redundant or less important parameters. We focus on post-training *structured* pruning, where entire channels or heads are removed so that the resulting model remains compatible with hardware acceleration. The central challenge is to reduce parameters while controlling the degradation in generation quality, especially for next-scale VAR models with long token sequences.

Layer-wise OBS formulation. Following Optimal Brain Surgeon (OBS), we decompose the global pruning problem into layer-wise subproblems and minimize the discrepancy between the original and pruned layer outputs in a local ℓ_2 sense. For the l-th layer with weight matrix W_l and input activation X_l, we consider the objective:

$$\min_{\hat{W}_l} \; \left\| W_l X_l - \hat{W}_l X_l \right\|_2^2, \qquad (2)$$

subject to a target pruning ratio on W_l. A second-order Taylor expansion around W_l yields a quadratic approximation of the loss in terms of the weight perturbation ΔW_l, with layer-wise Hessian:

$$H_l = 2X_l X_l^\top. \qquad (3)$$

For a scalar layer W_l, OBS iteratively identifies and removes the weight $w_p^\star$ that has the smallest effect on the objective output, and applies a closed-form compensation to

update remaining weights:

$$w_p^\star = \arg\min_{w_p} \; w_p^2 (H_l^{-1})_{p,p}, \qquad \delta_p = -\frac{w_p^\star}{(H_l^{-1})_{p,p}} (H_l^{-1})_{:,p}, \tag{4}$$

where $(H_l^{-1})_{p,p}$ and $(H_l^{-1})_{:,p}$ denote the p-th diagonal entry and column of H_l^{-1}, respectively. After pruning w_p to zero, δ_p is added to the remaining weights.

ExactOBS: row-wise formulation and inverse updates. To make OBS practical, we follow ExactOBS [9] and exploit two structural properties of the layer-wise objective. Dropping the layer index l for clarity, let $W \in \mathbb{R}^{d_{\text{row}} \times d_{\text{col}}}$ and $Y = WX$ be the dense layer output. The objective in Eq. (2) can be rewritten as a sum of row-wise errors:

$$\left\| WX - \hat{W}X \right\|_2^2 \; = \; \sum_{i=1}^{d_{\text{row}}} \left\| W_{i,:}X - \hat{W}_{i,:}X \right\|_2^2. \tag{5}$$

Removing a scalar entry $[W]_{i,j}$ only affects the error of the i-th output row $Y_{i,:}$; there is no Hessian interaction between different rows. Thus, each row can be treated as an independent least-squares problem with a $d_{\text{col}} \times d_{\text{col}}$ Hessian

$$H = 2XX^\top, \tag{6}$$

which is shared across all rows. This observation reduces the problem from a $d \times d$ Hessian to a much smaller $d_{\text{col}} \times d_{\text{col}}$ matrix.

The second ingredient of ExactOBS is an efficient update of the inverse Hessian when a single parameter p is removed. Let H be invertible with inverse H^{-1}, and let H_{-p} denote the principal submatrix obtained by removing row and column p from H. As shown in [9], the inverse H_{-p}^{-1} can be obtained directly from H^{-1} via a single Gaussian-elimination–style update:

$$H_{-p}^{-1} \; = \; \left(H^{-1} - \tfrac{1}{(H^{-1})_{p,p}} H_{:,p}^{-1} (H^{-1})_{p,:} \right)_{-p}, \tag{7}$$

where $(\cdot)_{-p}$ denotes removing row and column p. Intuitively, we eliminate the influence of parameter p from the inverse and then drop the corresponding row and column. This update costs $O(d_{\text{col}}^2)$ time and does not require recomputing any matrix inverses.

Structured OBS for VAR blocks. For next-scale VAR, we extend the above OBS formulation to operate on structured units in the pre-encode Transformer blocks at the last scale. Let $W^{(S)}$ denote a weight matrix in such a block and $H^{(S)}$ its Hessian. We adopt column pruning as the basic operation and treat groups of columns as structured units (e.g., attention heads or FFN channels). For a column index p, the structured OBS criterion and compensation can be written as

$$W_{:,p}^{(S)} = \arg\min_{W_{:,p}^{(S)}} \left\| W_{:,p}^{(S)} \right\|_2^2 (H^{(S)})_{p,p}^{-1}, \tag{8}$$

$$\Delta^{(S)} = -\frac{W_{:,p}^{(S)}}{(H^{(S)})_{p,p}^{-1}} (H^{(S)})_{p,:}^{-1}, \tag{9}$$

where $(H^{(S)})_{p,:}^{-1}$ is the p-th row of $(H^{(S)})^{-1}$ and $\Delta^{(S)}$ is a compensation matrix with the same shape as $W^{(S)}$. Following common practice, we prune attention blocks and FFNs as basic units: pruning columns in the attention output projection and FFN down-projection at the last scale reduces the number of heads and intermediate channels, thereby shrinking the model size.

Directly applying stepwise column pruning with OBS is still expensive for VAR due to the large intermediate dimensions and the interdependence between attention heads. We therefore adopt an *iterative unit-wise pruning-and-compensation scheme*: (i) define attention heads and FFN channels as structured units; (ii) estimate the OBS error for each unit using the Hessian statistics from the pre-encode calibration set (Sec. 3.3); (iii) prune the least important unit and apply the compensation update in Eq. (9); and (iv) repeat until the target sparsity is reached. For FFN layers, we further use a dynamic grouped strategy that prunes small groups of low-score channels at a time, starting with larger groups and gradually decreasing the group size as pruning progresses. This improves efficiency while keeping the solution close to the column-wise optimum. Empirically, this structured OBS scheme provides strong compression for VAR while maintaining high image generation quality.

3.3. Pre-Encode Calibration for Next-Scale VAR

OBS-guided pruning in a post-training setting relies critically on a calibration set: layer-wise Hessian estimates H_l are constructed from the activations induced by the calibration samples, and inaccurate calibration directly translates into suboptimal sensitivity estimates and poor training-free performance. For next-scale VAR, naive choices such as using tokens sampled from the model's own autoregressive generation pipeline (conditioned on prompts or partial inputs) lead to a severe mismatch between the calibration distribution and the real deployment regime.

To address this, we propose a *pre-encode calibration* strategy tailored to next-scale VAR. Instead of calibrating on model-generated tokens, we construct calibration samples by encoding real images through the exact VAE and residual pyramid used by VAR at training and inference time. Concretely, given a real image x, we first obtain a latent feature map $f \in \mathbb{R}^{h \times w \times C}$ via the VAE encoder. We then apply the VAR residual pyramid to decompose f into multi-scale residual feature maps, and quantize each scale using VAR's codebook and interpolation–lookup procedure to obtain the discrete token maps $(r_1, \ldots, r_K)$. Finally, we attach the same positional and scale embeddings as in VAR's training pipeline, yielding complete multi-scale to-

ken inputs that faithfully reflect the inference-time distribution.

We use this pre-encode calibration set to estimate Hessian statistics for the Transformer blocks at the last scale and to drive the OBS-based structured pruning described in Sec. 3.2. Because the calibration tokens are derived from real images and respect VAR's multi-scale residual hierarchy, the resulting Hessian estimates are substantially more stable and informative for sensitivity analysis.

3.4. Progressive Scale-Aware Distillation (PSAD)

Next-scale VAR inherently exhibits *scale-wise gradient imbalance*: Higher-resolution scales contain far more tokens than coarse scales, so unweighted objectives overemphasize fine scales and under-supervise coarse semantics. Pruning reduces model capacity and amplifies this bias, making global structure harder to preserve for the compressed model. We address this issue with *Progressive Scale-Aware Distillation* (PSAD), which couples a coarse-to-fine curriculum with scale-aware weighting to rebalance gradients across scales during post-pruning adaptation.

Setup. Let K be the number of scales and L the full sequence length after concatenating tokens from all scales. Scales are indexed by $i \in \{0, \dots, K-1\}$ with token boundaries $\mathbf{b} = [b_0, \dots, b_K]$ and per-scale counts $\mathbf{n} = [n_0, \dots, n_{K-1}]$, where we set

$$b_0 = 0, \quad b_K = L, \quad n_i = b_{i+1} - b_i, \quad \sum_{i=0}^{K-1} n_i = L. \tag{10}$$

Thus scale i occupies positions $\ell \in (b_i, b_{i+1}]$ in the flattened sequence. Let B denote the batch size. We use the original (unpruned) VAR as a teacher and the pruned VAR as a student. With temperature $\tau > 0$, teacher and student distributions at batch index b and position ℓ are

$$p_T^{(b,\ell)} = \mathrm{softmax}\big(\mathbf{z}_T^{(b)}[\ell,:]/\tau\big), \quad p_S^{(b,\ell)} = \mathrm{softmax}\big(\mathbf{z}_S^{(b)}[\ell,:]/\tau\big),$$

where $\mathbf{z}_T^{(b)}, \mathbf{z}_S^{(b)}$ denote teacher and student logits, respectively. The scale-level KL divergence (normalized by n_i) is defined as

$$\mathcal{L}_{\mathrm{KL}}^{(i)} = \frac{1}{B\, n_i} \sum_{b=1}^{B} \sum_{\ell=b_i+1}^{b_{i+1}} \mathrm{KL}\big(p_T^{(b,\ell)} \,\|\, p_S^{(b,\ell)}\big). \tag{11}$$

We include a factor τ^2 in the distillation loss below to compensate for the $1/\tau^2$ scaling of gradients with temperature.

1) Discrete stage-wise progression. We first describe a discrete coarse-to-fine schedule that progressively "unlocks" scales over training. Define the highest unlocked scale at optimization step t as

$$s(t) = \max\{\, i \in \{0, \dots, K-1\} \mid t \geq t_i \,\}, \quad \text{with } 0 = t_0 < t_1 < \cdots < t_{K-1} < T, \tag{12}$$

where t_i are pre-defined milestones and T is the total number of steps. Let $\gamma_i(t) = \mathbb{1}[\, i \leq s(t) \,]$ indicate whether scale i is active at step t. The progressive, scale-weighted distillation loss is then

$$\mathcal{L}_{\mathrm{PSAD}}(t) = \tau^2 \sum_{i=0}^{K-1} \gamma_i(t)\, w_i\, \mathcal{L}_{\mathrm{KL}}^{(i)}, \tag{13}$$

where w_i are per-scale weights (specified below). The task loss aggregates only the unlocked tokens:

$$\mathcal{L}_{\mathrm{task}}(t) = -\frac{1}{B\, b_{s(t)+1}} \sum_{b=1}^{B} \sum_{\ell=1}^{b_{s(t)+1}} \log p_S\big(x_\ell^{(b)} \mid x_{<\ell}^{(b)}, y^{(b)}\big), \tag{14}$$

where $x_\ell^{(b)}$ denotes the ground-truth token at position ℓ in sample b, and $y^{(b)}$ denotes the conditioning (e.g., class label). The total objective is

$$\mathcal{L}_{\mathrm{total}}(t) = \alpha\, \mathcal{L}_{\mathrm{task}}(t) + \beta\, \mathcal{L}_{\mathrm{PSAD}}(t), \tag{15}$$

with α and β balancing data loss and distillation.

2) Continuous soft progression. The hard indicator $\gamma_i(t)$ can lead to abrupt changes in the loss when a new scale is unlocked. To smooth these transitions, we replace $\gamma_i(t)$ with a continuous gate

$$\tilde{\gamma}_i(t) = \min\left\{1,\ \max\left\{0,\ \frac{t - t_i}{\Delta_i}\right\}\right\}, \tag{16}$$

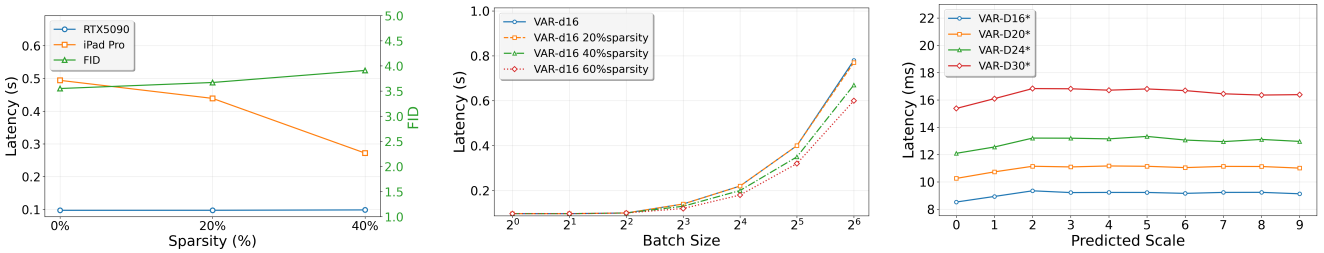
where $\Delta_i > 0$ controls the ramp-up duration for scale i. This yields a soft version of PSAD:

$$\mathcal{L}_{\mathrm{PSAD}}^{\mathrm{soft}}(t) = \tau^2 \sum_{i=0}^{K-1} \tilde{\gamma}_i(t)\, w_i\, \mathcal{L}_{\mathrm{KL}}^{(i)}. \tag{17}$$

In practice, we find that the soft schedule improves stability over the discrete variant, especially when the model is heavily pruned.

3) Progressive weights and gradient balancing. Token-length imbalance implies that, under a naïve uniform weighting, high-resolution scales with large n_i dominate the gradients. To counter this, we use monotonically decreasing scale weights, e.g.,

$$w_i = 2.0 - 0.2\, i, \quad i = 0, \dots, K-1, \tag{18}$$



(a) Latency–sparsity trade-off on RTX 5090 and iPad Pro (M4).

(b) Latency vs. batch size on RTX 5090 for different sparsity levels.

(c) Per-scale latency of pruned VAR models with different depths.

Figure 3. Latency behavior of EVAR under different sparsity and deployment settings. (a) As sparsity increases, single-image latency on the RTX 5090 remains almost unchanged, whereas latency on iPad Pro (M4) drops substantially while FID increases only mildly, indicating that EVAR makes single-image inference increasingly compute-bound on iOS with acceptable quality loss. (b) On the RTX 5090, dense and pruned VAR-d16 models show nearly identical latency at batch size 1, and only diverge at larger batch sizes, suggesting that single-image inference on the GPU is predominantly memory-bound. (c) For different VAR depths (VAR-D16*, VAR-D20*, VAR-D24*, VAR-D30*), per-scale latency for single-image inference is nearly flat across predicted scales, and pruning (indicated by *) does not introduce scale-dependent latency spikes, consistent with our width-oriented design.

which is consistent with the heuristic $w_i \propto 1/n_i$. In our experiments with $K = 10$, this yields positive and monotonically decreasing weights. The effective per-scale gradient magnitude scales roughly as

$$g_i(t) \;\propto\; (\gamma_i(t) \text{ or } \tilde{\gamma}_i(t)) \cdot w_i \cdot n_i \cdot \left\|\nabla_{\theta_S} \mathcal{L}_{\mathrm{KL}}^{(i)}\right\|.$$

The combination of progressive gates $(\gamma_i, \tilde{\gamma}_i)$ and decreasing w_i substantially reduces the dominance of high-resolution scales in the gradient, leading to more balanced multi-scale updates. In combination with the coarse-to-fine schedule, PSAD thus rebalances training signals across scales and significantly improves reconstruction fidelity and overall generative quality compared with conventional fine-tuning, particularly in the post-pruning setting.

4. Experimental Results

4.1. Experimental Setups

We evaluate EVAR on the ImageNet-1K [29] dataset using images of resolution 256×256, following exactly the same data preprocessing and augmentation pipeline as the original VAR model [44]. For a fair comparison, we adopt the VAR_d16 model as the base backbone on the ImageNet 256×256 conditional generation benchmark, and compare against state-of-the-art image generation model families.

We use class-balanced sampling with one image per class as the pre-encoded calibration set. Enlarging the calibration set yields no significant pruning gains, so we adopt "one per class" as the default.

After pruning, all models are fine-tuned for 40 epochs using AdamW with the same optimizer hyper-parameters, learning-rate schedule, and learning-rate / weight-decay annealing as in the original VAR training. Fine-tuning is performed on 8 NVIDIA 5090 GPUs. Finetuning the pruned model for one epoch takes slightly over one hour.

For edge deployment, the pruned models are further converted and deployed on an iPad Pro (M4), where we measure on-device latency for single-image generation.

4.2. Main Results

We compare our method with state-of-the-art pruning and compression approaches. Table 1 reports model size, parameter count, FLOPs, and storage reduction before and after pruning. Within the VAR family, we take VAR-d16 as our base model and report results for two pruned variants obtained with EVAR at 20% and 40% structured sparsity. At 20% pruning (EVAR, 270M parameters), our method maintains a FID of 3.67 compared to 3.55 for the dense VAR-d16, while reducing parameters from 310M / 10.97 GB to 270M. At 40% pruning (EVAR, 230M parameters), EVAR further compresses the model to 230M parameters with a FID of 3.91, which remains competitive given the substantial reduction in model size and decoding cost.

As summarized in Tab. 2, EVAR (ours) consistently outperforms a range of structured pruning baselines across sparsity levels and fine-tuning regimes. Under the most challenging setting of 40% sparsity without any fine-tuning, all methods suffer large quality drops, but EVAR still achieves the best FID/IS (64.19 / 19.52), whereas other methods degrade much more severely (e.g., FID $>$ 140 for OBA, Taylor, and LLM-Pruner). At 20% sparsity in the training-free regime, EVAR attains a substantially lower FID of 8.86 with strong precision/recall, while all baselines remain in a much worse range (FID 27.30–152.29), indicating that OBS-guided pruning with pre-encode calibration is particularly effective for moderate compression.

The row "Inference set" isolates the effect of calibration: it uses the same OBS-based pruning as EVAR but replaces our pre-encode calibration with tokens generated by VAR's own inference. Its performance is consistently worse than

Table 1. Generative performance on class-conditional ImageNet-256. "Steps" means the number of model inference to generate one image.

Type	Model	Parameters	Pruning Rate	Steps	FID↓	IS↑	Precision↑	Recall↑
VAR	VAR-d16 [44]	310M	–	10	3.55	274.4	0.84	0.51
	VAR-d20 [44]	600M	–	10	2.95	302.6	0.83	0.56
	VAR-d24 [44]	1.0B	–	10	2.33	312.9	0.82	0.59
AR	VQGAN-re [7]	1.4B	–	256	5.20	280.3	–	–
	RQ-Trans.-re [19]	3.8B	–	64	3.80	323.7	–	–
	LlamaGen-L [42]	343M	–	576	3.07	256.1	0.83	0.52
	LlamaGen-XL [42]	775M	–	576	2.62	244.1	0.80	0.57
	LlamaGen-XXL [42]	1.4B	–	576	2.62	244.1	0.80	0.57
	PAR-L [51]	343M	–	147	3.76	218.9	0.81	0.60
	PAR-XL [51]	775M	–	147	2.61	259.2	0.80	0.62
	PAR-XXL [51]	1.4B	–	147	2.35	263.2	0.80	0.62
	AiM-L [21]	350M	–	256	2.83	244.6	0.82	0.55
	AiM-XL [21]	763M	–	256	2.56	257.2	0.82	0.57
pruned VAR-d16	LLM-pruner [28]	230M	40%	10	4.21	53.92	0.81	0.50
	OBA [41]	230M	40%	10	4.19	53.43	0.83	0.47
	EVAR (ours)	270M	20%	10	3.67	57.78	0.81	0.51
	EVAR (ours)	230M	40%	10	3.91	57.23	0.81	0.51

Table 2. Comparison of pruning methods under different sparsity and fine-tuning regimes.

	40% sparsity, training-free				20% sparsity, training-free				40% sparsity, 1-epoch			
Method	FID	IS	Precision	Recall	FID	IS	Precision	Recall	FID	IS	Precision	Recall
EVAR (ours)	64.19	19.52	0.35	0.53	8.86	48.41	0.74	0.51	4.86	55.61	0.82	0.47
Inference set	138.91	6.39	0.09	0.07	56.87	19.35	0.36	0.57	9.62	44.91	0.77	0.46
LLM-pruner [28]	153.82	5.62	0.09	0.02	35.21	28.95	0.41	0.51	10.82	41.35	0.76	0.45
OBA [41]	142.82	6.35	0.12	0.04	30.54	29.48	0.45	0.57	9.85	44.35	0.78	0.46
Magnitude [12]	180.82	3.90	0.17	0.01	152.29	5.77	0.10	0.01	12.69	25.84	0.65	0.38
Taylor [30]	145.66	6.78	0.12	0.05	27.30	31.48	0.48	0.59	8.35	46.87	0.77	0.46

EVAR (e.g., FID 56.87 vs. 8.86 at 20% sparsity), showing that structured OBS compensation only works well when the Hessian is estimated from a precise dataset-aligned calibration set. After one epoch of post-pruning training, EVAR further improves to FID 4.86 at 40% sparsity, outperforming all baselines (best non-EVAR FID 8.35 for Taylor), and confirming that combining pre-encode calibration, OBS-guided structured pruning, and our distillation scheme yields the strongest overall trade-off between compression and generative quality.

Recent top-performing methods: LLM-Pruner and OBA. Since existing pruning methods for Transformers and large language models do not report results on VAR, we reimplement two structured pruning baselines on VAR-d16 using the public code: LLM-Pruner [28] and OBA [41]. For both methods, we prune VAR-d16 to a comparable 40% structured sparsity level and fine-tune for the same number of epochs as EVAR to ensure a fair comparison.

LLM-Pruner is applied with its default `param_mix` salience metric, which combines gradient and accumulated-gradient information to rank structured units. OBA is used in its structured setting, where sensitivity is estimated via Hessian–vector products over inter-layer connectivity and used to prune attention heads and FFN channels in VAR-d16. The corresponding results for LLM-Pruner, OBA, and our EVAR models are reported in the last block of Table 1.

4.3. Ablation Study

We ablate the distillation components in EVAR on ImageNet-1K single-image generation to quantify their effect on post-pruning recovery. Table 4 reports FID, iOS single-image latency, and parameter count for the dense VAR-d16 baseline, the training-free pruned model, and successive additions of fine-tuning and distillation components. Training-free pruning reduces latency from 494 ms

Device	Model	Compute units	Latency (ms)	#CPU ops	#GPU ops	#NPU ops
iPad Pro (M4)	EVAR	CPU+GPU	277 ± 2	10 (random_category)	8818	0
iPad Pro (M4)	EVAR	CPU+NPU	1090 ± 2	274	0	8554
iPad Pro (M4)	EVAR	All	465 ± 2	129	3103	5596

Table 3. Operator-level unit assignment in Core ML for a pruned VAR backbone (EVAR) on iPad Pro (M4). Frequent switches across CPU/GPU/NPU due to NE-unsupported ops inflate latency; pinning to CPU+GPU is fastest for single-image inference.

Method	**FID** ↓	**iOS (ms)** ↓	**Params (M)**
Dense	3.55	494	310
Pruned (training-free)	64.00	277	230
+ Fine-tuning (FT)	4.25	277	230
+ Vanilla KD	4.26	277	230
+ Scale-aware weights	3.97	277	230
+ Progressive (PSAD, ours)	**3.91**	277	230

Table 4. Ablation of the distillation components in EVAR. All post-pruning variants share the same pruned VAR-d16 backbone, hence identical latency and parameter counts under the same iOS deployment setting.

to 277 ms and parameters from 310M to 230M, but severely degrades FID (64.0). Plain fine-tuning and vanilla KD recover most of the quality (FID $\approx$ 4.25) while keeping the same latency/parameter budget, yet still underperform the dense model. Adding *scale-aware weights* further improves FID to 3.97, and enabling the full *Progressive Scale-Aware Distillation* (PSAD) yields the best result (FID 3.91), nearly closing the gap to the dense baseline under the same 277 ms / 230M on-device configuration.

4.4. Edge Deployment

Core ML conversion and runtime. For on-device evaluation, we convert the pruned VAR-d16 models from PyTorch to Core ML (`.mlmodel`) using `coremltools`, and run them on Apple silicon in a Swift-based single-image generation app on an iPad Pro (M4). The Core ML runtime performs per-operator scheduling and can *automatically switch compute units*—CPU, GPU, and Neural Engine (NE/NPU)—whenever an operator is unsupported on the currently selected unit.

Compute-unit behavior. Next-scale VAR contains several operators (e.g., positional embeddings and sampling-related ops) that are not supported on the NE, so enabling the NE causes frequent fallbacks and handoffs between CPU, GPU, and NE at batch size 1. Table 3 reports an operator-level breakdown for EVAR on an iPad Pro (M4) under three Core ML settings: `CPU+GPU`, `CPU+NPU`, and `All`. Although `CPU+NPU` assigns most operators to the NE, it incurs many CPU ops (274) and yields the highest latency (1090 ± 2 ms). The `All` setting mixes all three units (129 CPU ops, 3103 GPU ops, 5596 NPU ops) and is also slower (494 ± 2 ms) due to frequent cross-unit transitions. In contrast, pinning execution to `CPU+GPU` avoids NE-unsupported operators entirely, yields a simple two-unit schedule (10 CPU ops, 8818 GPU ops), and achieves the lowest latency (277 ± 2 ms) for single-image inference.

Latency analysis. Figure 3 summarizes the latency behavior of EVAR. In Fig. 3(a), increasing sparsity barely changes single-image latency on the RTX 5090, but significantly reduces latency on iPad Pro (M4) with only a mild FID increase, showing that EVAR is effective in the edge compute-bound regime. Fig. 3(b) shows that on the RTX 5090, dense and pruned VAR-d16 models have almost identical latency at batch size 1 and diverge only at larger batch sizes, indicating that single-image inference on GPU is largely memory-bound. Fig. 3(c) reports per-scale latency for different VAR depths (VAR-D16*, VAR-D20*, VAR-D24*, VAR-D30*); the curves are nearly flat across predicted scales and pruning (*) does not introduce scale-dependent latency spikes, consistent with our width-oriented design.

5. Conclusion

This paper introduces *EVAR*, a principled structured pruning framework for next-scale visual autoregressive (VAR) models. EVAR combines a pre-encode calibration pipeline with OBS-guided head/channel pruning and closed-form compensation, enabling training-free compression that respects the multi-scale conditioning structure of VAR. To further recover generative quality after pruning, we introduced Progressive Scale-Aware Distillation (PSAD), which rebalances supervision across scales and mitigates the gradient imbalance inherent to next-scale decoding. On ImageNet 256×256 single-image generation, EVAR achieves substantial reductions in parameters, memory footprint, and end-to-end latency while preserving competitive FID, IS, precision, and recall. In particular, a pruned VAR-d16 model on iPad Pro (M4) attains about $1.8\times$ speedup in single-image latency with only a modest FID increase. With a practical on-device deployment and evaluation pipeline for single-image, EVAR demonstrates that next-scale VAR can be compressed and executed efficiently on edge hardware, offering a practical step toward visual autoregressive image generation under tight resource constraints.

EVAR: Edge Visual Autoregressive Models via Principled Pruning

Supplementary Material

Appendix Overview

- **Section A:** Validates generality on LlamaGen-L-256, preserving metrics at 20% sparsity.
- **Section B:** Reports a 1.8× speedup on iOS devices, addressing CoreML operator constraints.
- **Section C:** Compares visual samples, confirming consistency between Linux and mobile deployments.

A. Generalization on LlamaGen

To further investigate the generality of our pruning method, we conducted experiments on another visual autoregressive image generation model, LlamaGen. Specifically, we selected LlamaGen-L-256 and applied 20% sparsity pruning, comparing it against VAR-d16 under the same 20% sparsity level. In processing the calibration set for LlamaGen, since it predicts next tokens, we encoded 256×256 images to obtain 256 tokens, which served as the pre-encoded calibration set to guide the structured OBS pruning. Other aspects remained consistent with the pruning process for the VAR model. Since LlamaGen employs next-token generation rather than next-scale generation, we did not apply our proposed Progressive Scale-Aware Distillation (PSAD) method; instead, we used standard fine-tuning. For fairness, we also applied standard fine-tuning to VAR-d16 as a baseline comparison.

The consolidated results are presented in Table 5. The findings indicate that our method generalizes effectively to other autoregressive architectures, reinforcing its broad applicability and robustness across diverse model families.

Table 5. Performance comparison with and without fine-tuning.

Model	FT	FID ↓	IS ↑	Prec. ↑	Rec. ↑
VAR-d16	w/ FT	3.81	60.43	0.84	0.51
	w/o FT	8.86	48.41	0.74	0.51
LlamaGen-L	w/ FT	3.57	258.3	0.82	0.50
	w/o FT	13.65	43.54	0.71	0.48

B. Deployment Details and Further Evaluation

During the conversion from PyTorch to CoreML, we encountered a limitation where the current version of coremltools does not support the bicubic interpolation operator. Consequently, we replaced it with bilinear sampling for mobile deployment. This operator switch initially caused the FID of the VAR-d16 model to increase from 3.55 to 9.98. However, after a brief fine-tuning phase of 20 epochs, the FID recovered to 4.34. While this represents a systematic increase of approximately 0.78 FID (and a corresponding rise in EVAR from 3.91 to 4.63) compared to the bicubic baseline, this degradation is strictly attributable to the toolchain limitation rather than the compression method. Crucially, the relative quality loss between the unpruned and pruned models—when both utilize bilinear sampling—remains within 10%. We anticipate this gap will vanish as coremltools support evolves.

We evaluated inference latency on iPad and iPhone devices using a robust trimmed mean protocol (20 runs after 5 warmups, excluding outliers). Results in Table 6 reveal two key findings. First, on iPads, our method maintains a consistent 1.8× speedup. Second, and most notably, both the full and 20% sparse models crashed on iPhones due to memory constraints (OOM). Only the 40% sparsity model operated successfully. This proves that high-ratio compression is not merely an accelerator but a strict prerequisite for deploying large generative models on memory-constrained edge devices.

Table 6. Inference Time Results on Different Devices (ms).

Device	full model	20% Sparsity	40% Sparsity
iPad Pro (M4)	494	439	277
iPad Pro (M2)	791	666	449
iPhone 16 PM	crashed	crashed	580
iPhone 12 Pro	crashed	crashed	1778

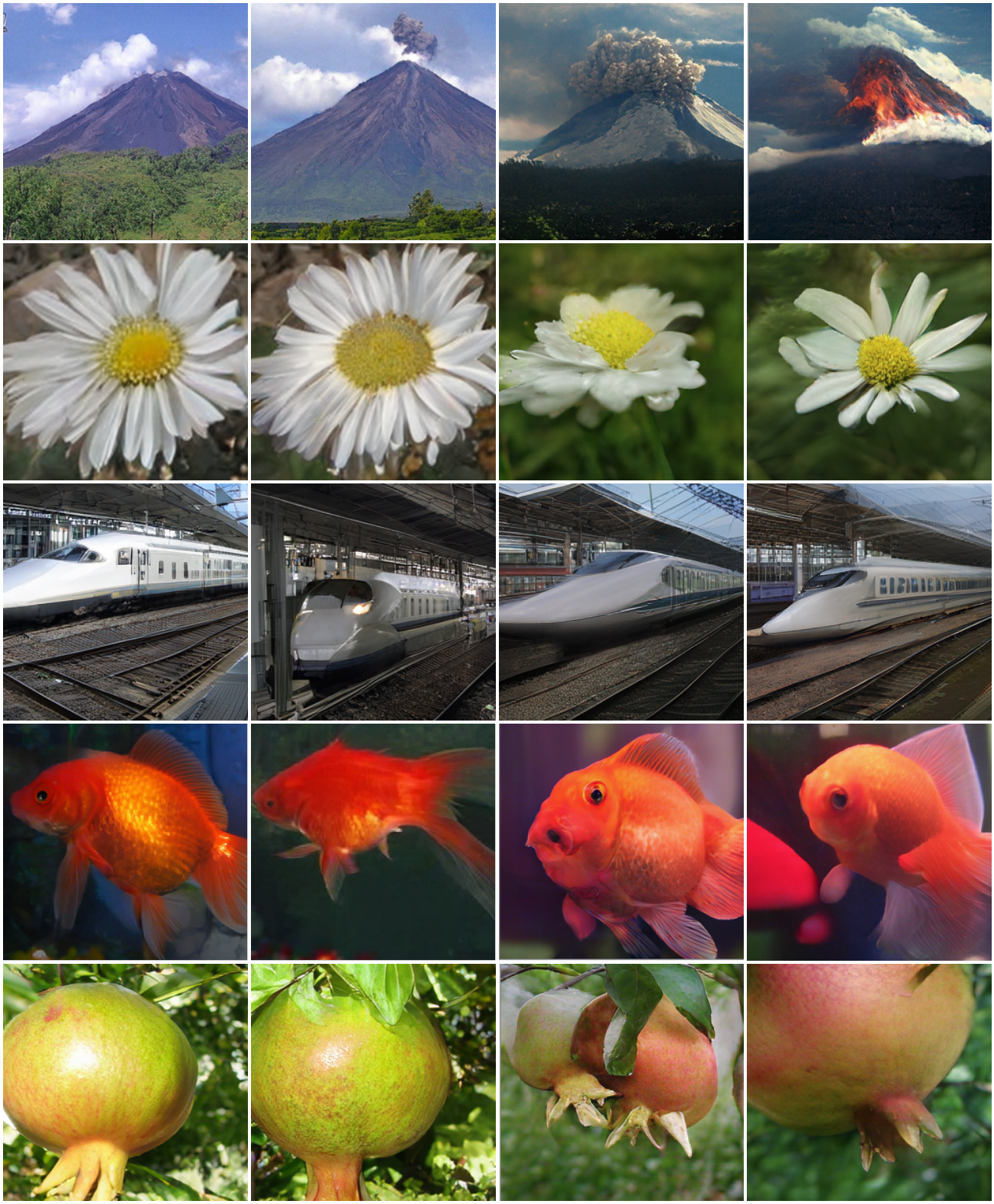
C. Visual Comparisons

Figure 4 presents a visual comparison between the unpruned VAR-d16 baseline and our pruned EVAR model (40% sparsity) under conditional generation settings. To demonstrate cross-platform consistency, we provide samples generated in two distinct environments: a standard Linux workstation using PyTorch and an actual on-device deployment via CoreML

It is important to note that the images generated on Apple devices exhibit minor visual deviations compared to the PyTorch baseline. These differences stem from the model conversion process and backend constraints. Most notably, since coremltools does not currently support bicubic sampling, we substituted it with bilinear sampling. Additionally, there are inherent discrepancies in operator implementation and fusion strategies between the PyTorch runtime and the CoreML backend.

Despite these constraints, the pruned EVAR model maintains high visual fidelity on mobile devices, closely mirroring the original model's capabilities.

Figure 4. Qualitative comparison of generated samples. We compare the unpruned VAR-d16 baseline executed on a Linux workstation (PyTorch) against our pruned EVAR model with 40% sparsity deployed on mobile devices (CoreML). Despite the aggressive compression and platform constraints, the mobile samples retain high visual fidelity comparable to the baseline.



(a) VAR-d16 on PyTorch (b) EVAR on PyTorch (c) VAR-d16 on CoreML (d) EVAR on CoreML

References

[1] Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. In *ICLR*, 2024. 2

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 2

[3] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. Maskgit: Masked generative image transformer. In *CVPR*, 2022. 2

[4] Tianyi Chen, Luming Liang, Tianyu Ding, Zhihui Zhu, and Ilya Zharkov. Otov2: Automatic, generic, user-friendly. In *ICLR*, 2023. 2

[5] Yongwei Chen, Yushi Lan, Shangchen Zhou, Tengfei Wang, and Xingang Pan. Sar3d: Autoregressive 3d object generation and understanding via multi-scale 3d vqvae. In *CVPR*, 2025. 1

[6] Zigeng Chen, Xinyin Ma, Gongfan Fang, and Xinchao Wang. Collaborative decoding makes visual auto-regressive modeling efficient. In *CVPR*, 2025. 1, 2

[7] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021. 1, 2, 7

[8] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. In *CVPR*, 2023. 2

[9] Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. In *NeurIPS*, 2022. 2, 4

[10] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *ICML*, 2023. 2

[11] Jiatao Gu, Yuyang Wang, Yizhe Zhang, Qihang Zhang, Dinghuai Zhang, Navdeep Jaitly, Joshua M. Susskind, and Shuangfei Zhai. Denoising autoregressive transformers for scalable text-to-image generation. In *ICLR*, 2025. 2

[12] et al Han, Song. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 2015. 2, 7

[13] Jian Han, Jinlai Liu, Yi Jiang, Bin Yan, Yuqi Zhang, Zehuan Yuan, Bingyue Peng, and Xiaobing Liu. Infinity: Scaling bitwise autoregressive modeling for high-resolution image synthesis. In *CVPR*, 2025. 1

[14] B. Hassibi, D.G. Stork, and G.J. Wolff. Optimal brain surgeon and general network pruning. In *NeurIPS*, 1993. 2

[15] Haoyu He, Jianfei Cai, Jing Liu, Zizheng Pan, Jing Zhang, Dacheng Tao, and Bohan Zhuang. Pruning self-attentions into convolutional layers in single path. *TPAMI*, 46(5):3910–3922, 2024. 2

[16] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017. 2

[17] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017. 2

[18] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *NeurIPS*, 1990. 2

[19] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. *arXiv preprint arXiv:2203.01941*, 2022. 2, 7

[20] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017. 2

[21] Haopeng Li, Jinyue Yang, Kexin Wang, Xuerui Qiu, Yuhong Chou, Xin Li, and Guoqi Li. Scalable autoregressive image generation with mamba. *arXiv preprint arXiv:2408.12245*, 2024. 1, 2, 7

[22] Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. *arXiv preprint arXiv:2406.11838*, 2024. 2

[23] Xiang Li, Kai Qiu, Hao Chen, Jason Kuen, Zhe Lin, Rita Singh, and Bhiksha Raj. Controlvar: Exploring controllable visual autoregressive modeling. *arXiv preprint arXiv:2406.09750*, 2024. 1

[24] Xiang Li, Kai Qiu, Hao Chen, Jason Kuen, Jiuxiang Gu, Bhiksha Raj, and Zhe Lin. Imagefolder: Autoregressive image generation with folded tokens. In *ICLR*, 2025. 1

[25] Liu Q Ling G, Wang Z. Slimgpt: Layer-wise structured pruning for large language models. In *NeurIPS*, 2024. 2

[26] Dongyang Liu, Shitian Zhao, Le Zhuo, Weifeng Lin, Yi Xin, Xinyue Li, Qi Qin, Yu Qiao, Hongsheng Li, and Peng Gao. Lumina-mgpt: Illuminate flexible photorealistic text-to-image generation with multimodal generative pretraining. *arXiv preprint arXiv:2408.02657*, 2024. 2

[27] Enshu Liu, Xuefei Ning, Yu Wang, and Zinan Lin. Distilled decoding 1: One-step sampling of image auto-regressive models with flow matching. In *ICLR*, 2025. 1

[28] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *NeurIPS*, 2024. 2, 7

[29] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Imagenet: A large-scale hierarchical image database. *CVPR*, 2009. 6

[30] Tyree S Molchanov P, Mallya A. Importance estimation for neural network pruning. In *CVPR*, 2019. 2, 7

[31] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. In *NeurIPS*, 2016. 2

[32] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 2

[33] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini

Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022. 2

[34] William Peebles and Saining Xie. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022. 2

[35] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. *Semantic Scholar*, 2018. 2

[36] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 2

[37] Sucheng Ren, Yaodong Yu, Nataniel Ruiz, Feng Wang, Alan Yuille, and Cihang Xie. M-var: Decoupled scale-wise autoregressive modeling for high-quality image generation. *arXiv preprint arXiv:2411.10433*, 2024. 1

[38] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *arXiv preprint arXiv:abs/2112.10752*, 2021. 2

[39] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. In *NeurIPS*, 2024. 2

[40] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. In *ICLR*, 2024. 2

[41] Mingyuan Sun, Zheng Fang, Jiaxu Wang, Junjie Jiang, Delei Kong, Chenming Hu, Yuetong Fang, and Renjing Xu. Optimal brain apoptosis. In *ICLR*, 2025. 2, 7

[42] Peize Sun, Yi Jiang, Shoufa Chen, Shilong Zhang, Bingyue Peng, Ping Luo, and Zehuan Yuan. Autoregressive model beats diffusion: Llama for scalable image generation. *arXiv preprint arXiv:2406.06525*, 2024. 1, 2, 7

[43] Haotian Tang, Yecheng Wu, Shang Yang, Enze Xie, Junsong Chen, Junyu Chen, Zhuoyang Zhang, Han Cai, Yao Lu, and Song Han. Hart: Efficient visual generation with hybrid autoregressive transformer. In *ICLR*, 2025. 1

[44] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. In *NeurIPS*, 2024. 1, 3, 6, 7

[45] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016. 2

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 2

[47] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *ICLR*, 2021. 2

[48] Huan Wang, Yulun Zhang, Can Qin, Luc Van Gool, and Yun Fu. Global aligned structured sparsity learning for efficient image super-resolution. *TPAMI*, 45(9):10974–10989, 2023. 2

[49] Jinhong Wang, Jian Liu, Dongqi Tang, Weiqiang Wang, Wentong Li, Danny Chen, Jintai Chen, and Jian Wu. Scalable autoregressive monocular depth estimation. In *CVPR*, 2025. 1

[50] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. PVT v2: Improved baselines with pyramid vision transformer. *Comput. Vis. Media*, 8(3):415–424, 2022. 2

[51] Yuqing Wang, Shuhuai Ren, Zhijie Lin, Yujin Han, Haoyuan Guo, Zhenheng Yang, Difan Zou, Jiashi Feng, and Xihui Liu. Parallelized autoregressive visual generation. *arXiv preprint arXiv:2412.15119*, 2024. 1, 2, 7

[52] Ma Xiaoxiao, Zhou Mohan, Liang Tao, Bai Yalong, Zhao Tiejun, Li Biye, Chen Huaian, and Jin Yi. Star: Scale-wise text-conditioned autoregressive image generation. *arXiv preprint arXiv:2406.10797*, 2024. 1

[53] Rui Xie, Tianchen Zhao, Zhihang Yuan, Rui Wan, Wenxi Gao, Zhenhua Zhu, Xuefei Ning, and Yu Wang. Litevar: Compressing visual autoregressive modelling with efficient attention and quantization. *arXiv preprint arXiv:2411.17178*, 2024.

[54] Qian Zhang, Xiangzi Dai, Ninghua Yang, Xiang An, Ziyong Feng, and Xingyu Ren. Var-clip: Text-to-image generator with visual auto-regressive modeling. *arXiv preprint arXiv:2408.01181*, 2024. 1

[55] Chunting Zhou, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. Transfusion: Predict the next token and diffuse images with one multi-modal model. *arXiv preprint arXiv:2408.11039*, 2024. 2