POOLING REPRESENTATION AUTOENCODERS FOR EFFICIENT DIFFUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Representation autoencoders (RAEs) generate images from pretrained visual features, but their dense token grids make generative modeling expensive. Motivated by local feature correlations, we introduce PoolDINO, a learned affine pooling operator that merges neighboring tokens. Training the pooling operator jointly with the RGB decoder preserves the standard two-stage RAE procedure without a separate feature autoencoder. On ImageNet-256, $4\times$ token compression retains comparable generation quality under internal guidance, while $16\times$ compression trades some quality for greater efficiency. At a fixed budget of 100 sampling steps, latent-sampling throughput increases by $3.7\times$ and $9.0\times$, respectively, relative to the unpooled baseline. Classification and dense prediction evaluations show that comparable guided generation quality can coexist with weaker performance on other tasks.

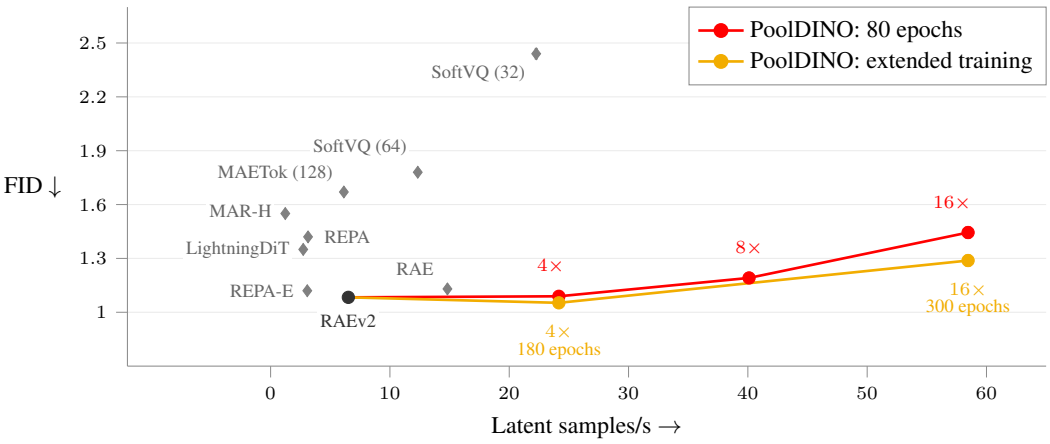

Figure 1: Generation quality versus latent-sampling throughput on an NVIDIA H100 NVL at batch size 128. PoolDINO and the RAEv2 reference use Internal Guidance (IG) with 100 Euler steps. Red points use 80 training epochs; gold points use extended training (180 epochs for 2×2 and 300 for 4×4). Diamonds pair published FIDs with measured sampling rates. Appendix B details the measurement protocols and shows that reducing to 50 sampling steps doubles throughput while maintaining comparable generation quality.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Pretrained self-supervised visual features have become a powerful foundation for image generation. REPA shows that using these features as alignment targets substantially accelerates diffusion training and improves sample quality (Yu et al., 2025). Representation autoencoders (RAEs) use them directly as the generative latent space within a two-stage framework: first training an image decoder on frozen vision-encoder features, and then training a flow-matching model to generate these features (Zheng et al., 2026). Although earlier work identified optimization difficulties in high-dimensional latent spaces (Yao et al., 2025), RAEs successfully enable efficient generation directly from these features, with RAEv2 yielding further gains (Singh et al., 2026b).

However, the spatial resolution of these representations translates into long token sequences that are computationally expensive to model with Transformers (Vaswani et al., 2017). The iREPA analysis demonstrates that nearby tokens in these feature maps are strongly correlated (Singh et al., 2026a). Because these tokens are also high-dimensional, we hypothesize that a single pooled token possesses enough capacity to summarize its local neighborhood, thereby reducing both the token count and the associated memory and compute requirements of the generative Transformer.

Building on RAEv2, we introduce PoolDINO, a learned affine operator that pools neighboring encoder patches into a single token. Pooling reduces the sequence length while retaining an explicit coarse two-dimensional grid. We train the pooling operator jointly with the RGB decoder while keeping the vision encoder frozen. The decoder's perceptual and adversarial objectives favor perceptually faithful reconstruction rather than exact recovery of every image detail (Zheng et al., 2026). Learning a pooling operator under these objectives allows reconstruction requirements to guide compression. We then freeze the pooling operator and decoder and train the generator on the compressed representations. By learning compression jointly with RGB reconstruction, PoolDINO preserves the two-stage RAEv2 training pipeline.

By varying the compression rate, we establish a controlled trade-off between generation quality and sampling throughput (Figure 1). At $4\times$ compression, we retain generation quality comparable to the uncompressed reference, while more aggressive scales trade fidelity for significant efficiency gains. We also examine the downstream transferability of the pooled representations through operator analysis, image classification, semantic segmentation, and monocular depth estimation. Classification accuracy decreases despite comparable guided generation quality at $4\times$, and pooling learned for RGB reconstruction does not consistently outperform average pooling on the dense prediction tasks.

Our contributions are:

- **Method:** We introduce PoolDINO, a local affine pooling operator trained jointly with an RGB decoder to compress frozen vision-encoder representations (Section 4).
- **Generation:** We characterize the trade-off between token compression, guided generation quality, and sampling throughput on ImageNet-256 (Section 5.2; Appendix B).
- **Analysis:** We assess the semantic content using operator analysis, classification, semantic segmentation, and monocular depth estimation (Section 5.3; Appendix F).

2 RELATED WORK

Pretrained representations for generation. REPA aligns a diffusion model's hidden states with pretrained visual features to accelerate training and improve generation quality (Yu et al., 2025). iREPA identifies spatial self-similarity in these features as a stronger predictor of alignment effectiveness than classification accuracy (Singh et al., 2026a). Pretrained features can also be used to guide the latent space itself: VA-VAE aligns variational autoencoder (VAE; Kingma & Welling 2014) latents with vision-encoder features during tokenizer training (Yao et al., 2025), while REPA-E uses representation alignment to jointly tune the VAE and diffusion model (Leng et al., 2025).

RAEs use frozen vision-encoder features directly as generative latents, training an image decoder to reconstruct RGB images from them (Zheng et al., 2026). RAEv2 combines features from multiple encoder layers to improve reconstruction and introduces further refinements to generator training (Singh et al., 2026b).

Compressing self-supervised features. FAE reduces the channel dimension of RAE latents using a single attention layer and a linear projection (Gao et al., 2025). However, this requires an additional training stage to learn the feature autoencoder. Moreover, because FAE strictly maintains the original spatial token count, it fails to alleviate the computational bottleneck in the generation stage.

FlatDINO addresses spatial redundancy with a Transformer autoencoder that compresses dense DINOv2 features into a shorter token sequence (Calvo-González & Fleuret, 2026). Like FAE, it introduces an extra training stage for feature compression. Furthermore, despite its complex architecture, most of its latent tokens learn to compress fixed spatial chunks independent of the image content. In practice, FlatDINO yields a worse trade-off between throughput and generation quality than PoolDINO (Figure 8).

Motivated by strong correlations between neighboring features, PoolDINO employs explicit spatial pooling, compressing each local window with a simple affine map trained jointly with the RGB decoder. This directly optimizes compression for image reconstruction, avoiding the separate feature autoencoders required by FAE and FlatDINO while preserving the two-stage RAE training pipeline.

3 PRELIMINARIES

3.1 FLOW MATCHING

Given samples from an unknown data distribution p_{data}, the goal of generative modeling is to produce new samples that follow the same distribution. Flow matching approaches this task by learning a time-dependent velocity field that transforms samples from a simple distribution, typically Gaussian noise, into data (Lipman et al., 2023).

To construct training examples, we independently sample $x \sim p_{\text{data}}$ and $\epsilon \sim \mathcal{N}(0, I)$, and interpolate between them:

$$x_t = (1-t)\epsilon + tx, \qquad t \in [0, 1]. \tag{1}$$

Here, $t = 0$ corresponds to noise and $t = 1$ to clean data. For each sampled pair, this path has constant velocity $x - \epsilon$.

A model $v_\theta(x_t, t)$ learns to predict this velocity by minimizing

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{x,\epsilon,t}\left[\|v_\theta(x_t, t) - (x - \epsilon)\|_2^2\right]. \tag{2}$$

Although each training target depends on a particular data–noise pair, the optimal predictor averages the velocities compatible with the observed x_t. This yields a velocity field that transports the corresponding distributions, without requiring numerical integration during training.

To generate a sample, we initialize $x_0 \sim \mathcal{N}(0, I)$ and numerically integrate $\mathrm{d}x_t/\mathrm{d}t = v_\theta(x_t, t)$ from $t = 0$ to $t = 1$. The same formulation applies to latent representations in place of images. The model can also predict the clean sample $f_\theta(x_t, t)$, with the velocity obtained analytically as $v_\theta(x_t, t) = (f_\theta(x_t, t) - x_t)/(1 - t)$ for $t < 1$.

3.2 REPRESENTATION AUTOENCODERS

The RAE framework uses a pretrained vision encoder E to map an image $x \in \mathbb{R}^{H\times W\times 3}$ to a latent representation $z \in \mathbb{R}^{h\times w\times D}$, where h, w define the spatial grid size and D is the token dimensionality. An image decoder D_θ is then trained to recover x directly from z.

While the original RAE extracts z from the final block K of the encoder, $z = E_K(x)$, RAEv2 defines the latent space as an average over a set of intermediate layers $\mathcal{L}$ to balance the fine-grained spatial structure of earlier layers with the high-level semantic context of deeper layers:

$$z = \frac{1}{|\mathcal{L}|}\sum_{\ell\in\mathcal{L}} \mathrm{LN}(E_\ell(x)),$$

where $E_\ell(x)$ denotes the activations after block ℓ, and LN is a parameter-free layer normalization. Following RAEv2, we use a DINOv3-L/16 encoder (Siméoni et al., 2026) with $\mathcal{L} = \{11, 13, 15, 17, 19, 21, 23\}$.

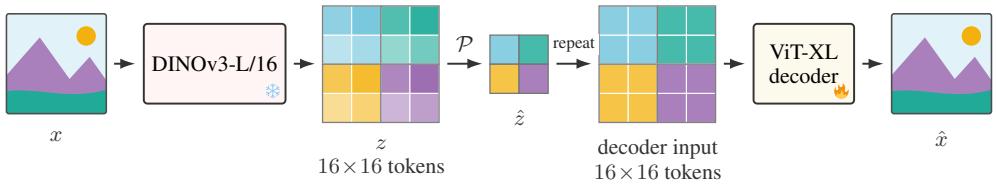

Figure 2: Training the pooled image decoder. A frozen DINOv3-L/16 produces a dense patch field. The learned projection $\mathcal{P}$ maps each non-overlapping pooling window to one token, which is repeated over its original window before ViT-XL reconstructs the image. Shades distinguish source patches; uniform colors within the repeated blocks denote identical token copies. A 4×4 field with 2×2 pooling windows is shown for clarity.

4 METHOD

4.1 IMAGE DECODING

Two properties of continuous semantic latents make them well suited to spatial compression. First, nearby vision-encoder features exhibit strong local correlation (Singh et al., 2026a). Second, pooling preserves the high-dimensional channel capacity D of the original tokens, providing each pooled token with sufficient bandwidth to summarize the features of several constituent patches. Because the number of scalar elements in the uncompressed latent field z can be comparable to that in the raw image x, we hypothesize that a learned spatial pooling operator can heavily compress this token grid while preserving the essential semantic structure required for high-quality image decoding.

Given the RAEv2 latent $z \in \mathbb{R}^{h \times w \times D}$, we partition it into non-overlapping windows of size $p_y \times p_x$ and map each window to a single token:

$$\hat{z}_{i,j} = \mathcal{P}(z[ip_y : (i+1)p_y,\ jp_x : (j+1)p_x,\ :]) \,. \tag{3}$$

Here, the colon denotes array slicing, and $\mathcal{P} : \mathbb{R}^{p_y \times p_x \times D} \to \mathbb{R}^D$ is applied independently to each window. Assuming that h and w are divisible by p_y and p_x, respectively, this gives the pooled latent $\hat{z} \in \mathbb{R}^{\frac{h}{p_y} \times \frac{w}{p_x} \times D}$.

We use learned affine pooling: $\mathcal{P}$ is a shared affine map with weight $W \in \mathbb{R}^{D \times mD}$ and bias $b \in \mathbb{R}^D$, where $m = p_x p_y$, applied to the concatenated features of each window. This is equivalent to a convolution with kernel size and stride both equal to (p_y, p_x). Before RGB decoding, we repeat each pooled token over its original window. We compare this learned pooling against fixed average and max pooling.

A ViT-XL decoder then learns to reconstruct x from the repeated pooled field. Repetition restores the original grid, keeping the decoder architecture and input sequence length fixed across pooling geometries to control for decoder compute.

4.2 IMAGE GENERATION

With the tokenizer and RGB decoder frozen, we train one class-conditional diffusion Transformer (DiT^{DH}-XL; Peebles & Xie 2023; Singh et al. 2026b) for each pooling geometry. The generator operates directly on the corresponding pooled latent grid,

$$z = E(x), \qquad \hat{z} = \mathcal{P}(z), \qquad \bar{z} = \frac{\hat{z} - \mu_{\hat{z}}}{\sigma_{\hat{z}}}, \tag{4}$$

where $\mu_{\hat{z}}$ and $\sigma_{\hat{z}}$ are channel-wise statistics. We use the RAEv2 architecture and transport objective, adapting the spatial input grid to each pooling geometry. Unless otherwise specified, each model is trained for 80 epochs with a global batch size of 1,024, using a hybrid Muon–AdamW optimizer (Jordan et al., 2024; Loshchilov & Hutter, 2019) and a base learning rate of 2×10^{-4}. The learning rate is held constant through epoch 25 and decayed linearly to 2×10^{-5} by epoch 50. We drop the class condition with probability 0.1 to enable classifier-free guidance. Full optimization details are given in Table 7. We also use the internal guidance (IG; Zhou et al. 2026) mechanism, which trains an

intermediate generator layer to predict the clean latent. During sampling, extrapolating from this weaker intermediate prediction to the full-depth prediction provides a strong guidance signal without requiring a separate unconditional evaluation.

The training objective contains the usual flow-matching loss and two auxiliary losses applied to the hidden states after the eighth DiT block:

$$\mathcal{L} = \mathcal{L}_{\mathrm{FM}} + \mathcal{L}_{\mathrm{IG}} + 0.5\,\mathcal{L}^{E}_{\mathrm{rec}}. \tag{5}$$

Here $\mathcal{L}_{\mathrm{FM}}$ is the flow-matching loss of the full generator (using an x-prediction objective; Li & He 2026), and $\mathcal{L}_{\mathrm{IG}}$ applies the same objective to the intermediate IG prediction. A second auxiliary head reconstructs the original dense encoder patch field $z = E(x)$ using mean-squared error ($\mathcal{L}^{E}_{\mathrm{rec}}$). Both auxiliary heads are trained jointly for every model. Appendix A gives the full training diagram.

During sampling, let $\bar{z}_{\mathrm{full}}$ denote the full-depth prediction and $\bar{z}_{\mathrm{int}}$ the intermediate prediction. IG extrapolates between them:

$$\bar{z}_{\mathrm{IG}} = \bar{z}_{\mathrm{int}} + s_{\mathrm{IG}}\,(\bar{z}_{\mathrm{full}} - \bar{z}_{\mathrm{int}}), \tag{6}$$

where $s_{\mathrm{IG}} = 1$ recovers the original prediction. We evaluate IG both alone (used in Figure 1) and combined with standard classifier-free guidance (CFG; Ho & Salimans 2021), whose scale is s_{CFG}. Scales are selected separately for each model. Appendix E gives full ablation results, including a comparison between IG and guidance derived from the encoder-reconstruction head.

After sampling, we undo the latent normalization, repeat each pooled token over its original spatial window, and apply the frozen RGB decoder:

$$x_{\mathrm{gen}} = D_\theta(\mathrm{repeat}\,(\sigma_{\hat{z}}\,\bar{z}_{\mathrm{sample}} + \mu_{\hat{z}})). \tag{7}$$

Repetition restores the decoder's 16×16 input grid but introduces no additional information; we do this to strictly control for decoder FLOPs across all pooling geometries. Appendix A lists the latent grids, architecture, and optimization settings.

5 Results

We evaluate reconstruction and class-conditional generation on ImageNet-1K (Russakovsky et al., 2015) at 256×256 resolution.

5.1 Reconstruction

Learning the pooling operator preserves reconstruction quality more effectively than parameter-free pooling (Table 1). At $4\times$ compression, learned pooling reaches a reconstruction Fréchet Inception Distance (rFID; Heusel et al. 2017) of 0.36, close to the unpooled reference at 0.32 and lower than the average pooling control at 0.65. At $16\times$, its rFID rises to 0.41, compared with 1.81 for average pooling and 1.72 for max pooling. PSNR and LPIPS also favor learned pooling over both parameter-free alternatives.

Table 1: Reconstruction on the ImageNet-1K 256×256 validation set, grouped by pooling method. All RGB decoders process 256 tokens after repetition.

Spatial pooling	Tokens	rFID ↓	sFID ↓	PSNR ↑	LPIPS ↓
RAEv2 baseline	256	0.32	2.29	22.74	0.15
Learned 2×2	64	0.36	2.61	22.25	0.17
Learned 2×4	32	0.39	2.76	21.88	0.18
Learned 4×2	32	0.39	2.82	21.85	0.18
Learned 4×4	16	0.41	2.99	21.45	0.19
Average 2×2	64	0.65	4.97	19.58	0.25
Average 4×4	16	1.81	8.65	16.64	0.37
Max 2×2	64	0.71	5.17	19.17	0.26
Max 4×4	16	1.72	8.31	16.22	0.38



Figure 3: Curated samples across spatial compression levels, with all generators trained for 80 epochs. Samples are selected independently across columns. Sampling uses 100 Euler steps and each model's selected IG-only scale. Appendix G provides randomly selected samples without quality filtering.

5.2 Image Generation

Table 2 shows that $4\times$ compression retains comparable image generation quality with IG alone (1.09 FID versus the uncompressed baseline's 1.08). While $16\times$ compression trades some fidelity for greater efficiency by reaching 1.44 FID, adding CFG further improves the lowest observed FID across all models. This combined guidance yields the largest improvements at $16\times$ compression, though it requires an additional unconditional forward pass per sampling step. Appendix E gives the full comparison with CFG alone and encoder-reconstruction guidance, including the selected scales and complete sweeps.

Average pooling baseline. We compare the learned affine mapping against a simpler pooling strategy by training a 2×2 average-pooled generator for 80 epochs. While it achieves stronger unguided generation (1.70 FID) than its learned counterpart (3.00 FID), it responds less effectively to IG (1.29 FID) and fails to match the uncompressed baseline.

Table 2: ImageNet-256 generation at 100 Euler steps. Models train for 80 epochs. We report generation FID and Inception Score (IS; Salimans et al. 2016). Guided columns report the lowest-FID settings from our guidance scale sweeps; the corresponding scales are in Table 16. Best and second-best metrics in each column are bold and underlined.

		Unguided		IG		IG + CFG	
Spatial pooling	Tokens	FID ↓	IS ↑	FID ↓	IS ↑	FID ↓	IS ↑
RAEv2 baseline	256	**1.53**	<u>226.63</u>	**1.08**	<u>262.00</u>	<u>1.07</u>	267.15
Learned 2×2	64	3.00	184.49	<u>1.09</u>	250.36	**1.07**	268.50
Learned 2×4	32	4.91	161.42	1.19	249.58	1.16	273.06
Learned 4×2	32	4.97	159.67	1.21	249.96	1.19	<u>273.98</u>
Learned 4×4	16	8.16	132.67	1.44	247.07	1.35	**282.29**
Average 2×2	64	<u>1.70</u>	**232.55**	1.29	**268.08**	–	–

Extended training. The lower per-update cost of compressed generators allows longer training within the compute budget of the unpooled baseline. We train the 2×2 and 4×4 generators for 180 and 300 epochs, respectively (Appendix A.1). Table 3 shows that extended training improves generation quality at both compression rates. Figure 4 compares samples from the 80-epoch and extended-training checkpoints, which are also included in Figure 1.

Fewer sampling steps. Reducing sampling from 100 to 50 steps maintains comparable generation quality while providing a further twofold increase in latent-sampling throughput. Appendix B.1 provides the full comparison.

Table 3: Effect of extended training, evaluated with 100 Euler steps. The unpooled 80-epoch model is included as a reference. IG columns report the lowest-FID tested setting and its IS; bold marks the stronger result within each pooling window.

		Unguided		IG	
Pooling	Epochs	FID ↓	IS ↑	FID ↓	IS ↑
1×1 (unpooled)	80	1.53	226.63	1.08	262.00
2×2	80	3.00	184.49	1.09	250.36
2×2	180	**2.78**	**192.37**	**1.05**	**272.00**
4×4	80	8.16	132.67	1.44	247.07
4×4	300	**7.19**	**146.13**	**1.29**	**258.04**

Figure 3 shows generated samples across compression levels. Appendix G provides randomly selected samples from the unpooled reference, the compressed models, and both extended-training checkpoints.

5.3 Classification and dense prediction

Beyond reconstruction and generation, we ask what information remains accessible in the compressed representations. We evaluate both global semantic information through image classification and localized information through semantic segmentation and monocular depth estimation.

Image classification and operator analysis. We evaluate global semantic content on ImageNet classification by spatially averaging the pooled field $\hat{z}$ into one feature vector per image. Features from learned pooling yield lower accuracy than all three simpler baselines at every compression rate on both linear probing and k-NN evaluations (Table 4; Appendix C). To understand this difference, we examine the operator's subspace: the learned operator retains a different subspace from average pooling and local PCA. Its alignment with both decreases as compression increases, and it preserves less total variance than PCA at the same output dimension (Appendix F). These measurements characterize the learned representation, but do not establish which preserved directions account for the generation results.

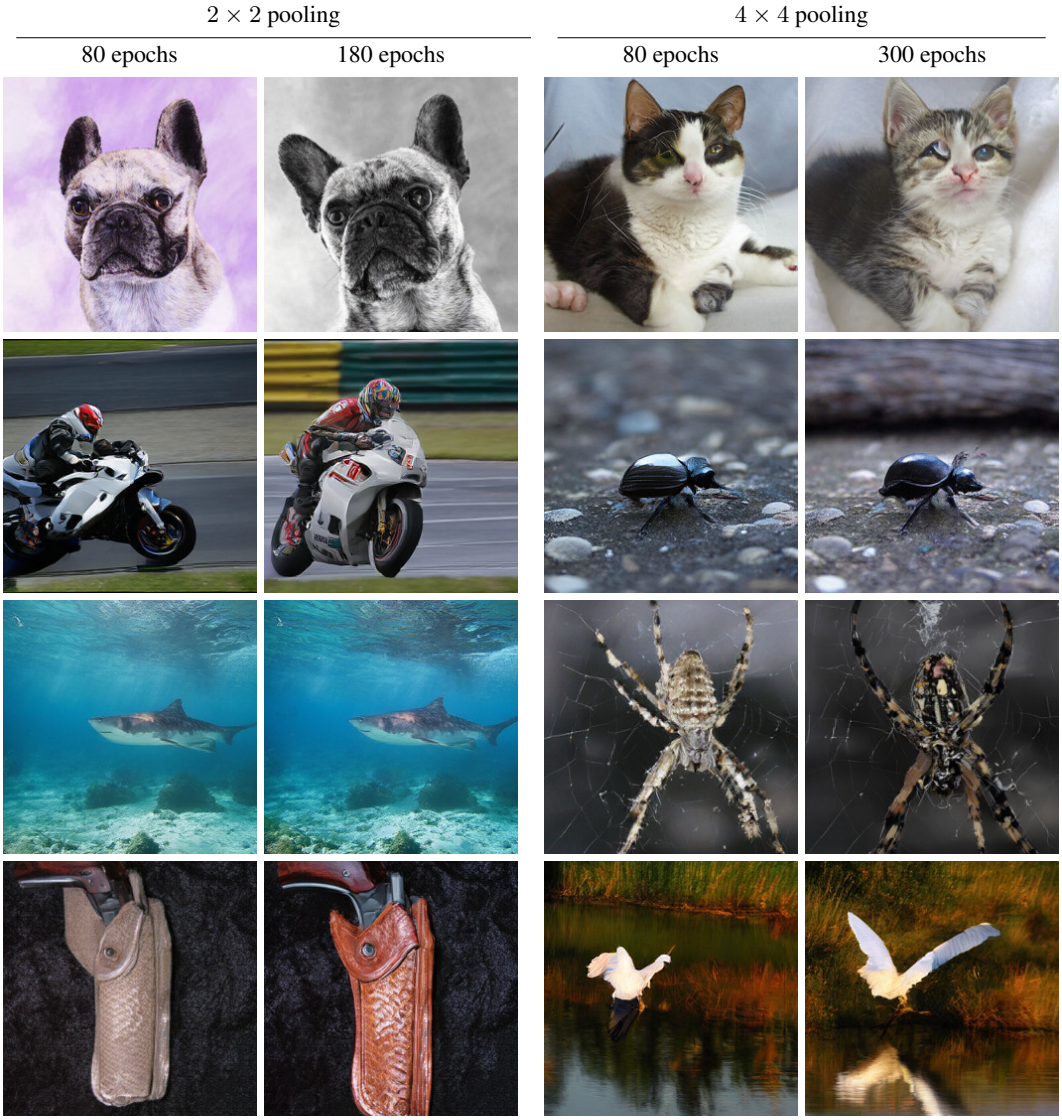

Figure 4: Curated comparisons of 80-epoch and extended training. Class conditions and initial noise are matched within each pair. All samples use 100 Euler steps and IG alone: scales 1.75/2.00 for the 80/180-epoch 2×2 models, and 2.75 for both 4×4 models.

Dense prediction. Because spatial pooling reduces resolution, we test whether the compressed token grid still preserves the localized semantic and geometric details required for pixel-level tasks. We evaluate this on ADE20K (Zhou et al., 2017) semantic segmentation and NYUv2 (Silberman et al., 2012) monocular depth estimation by freezing the encoder and pooling operator and training a randomly initialized ViT-XL task model for each representation (Table 4). We report mean intersection over union (mIoU) for segmentation and absolute relative error (AbsRel) for depth. At $4\times$ compression, average pooling performs better on both tasks. At $16\times$, learned pooling gives higher segmentation mIoU, while average pooling retains lower depth error. Thus, the comparable guided generation quality achieved at $4\times$ coexists with weaker transfer to dense prediction. These results do not establish a consistent transfer advantage for pooling learned through RGB reconstruction. Future work should test whether learning $\mathcal{P}$ separately for each downstream task yields more suitable representations. Appendix D gives the complete metrics and results with RGB-decoder initialization.

Table 4: Classification and dense prediction from frozen representations. Dense task models train from scratch. Bold marks the stronger pooling rule at each compression rate.

		ImageNet-1K	ADE20K	NYUv2
Spatial pooling	Window	Linear top-1 (%) ↑	mIoU (%) ↑	AbsRel ↓
Unpooled	1×1	85.31	46.86	0.0817
Learned	2×2	83.47	39.38	0.0895
Average	2×2	**85.33**	**42.59**	**0.0869**
Learned	4×4	80.80	**38.61**	0.1022
Average	4×4	**85.33**	36.76	**0.0997**

6 Conclusion

We introduced PoolDINO, a learned spatial pooling operator that compresses representation autoencoder latents for efficient generative modeling. By training a local affine mapping jointly with an RGB decoder, PoolDINO avoids the complex feature autoencoders required by prior methods and fully preserves the standard two-stage RAE training pipeline.

Our evaluations demonstrate a trade-off between computational cost and generation quality. At $4\times$ compression, PoolDINO substantially increases sampling throughput while retaining a guided FID highly competitive with the uncompressed baseline. More aggressive $16\times$ compression enables substantial efficiency gains, trading visual fidelity for even higher throughput.

Finally, we evaluated transfer to classification and dense prediction. Learned pooling yields lower k-NN and linear-probe accuracy than average pooling, local PCA, and random projection, and does not consistently outperform average pooling on dense prediction tasks. These results show that strong guided generation quality does not guarantee equally strong downstream performance. Future work should investigate pooling objectives tailored to downstream tasks or jointly optimized for generation and transfer.

Limitations. Our generation experiments are restricted to ImageNet-256 and a single autoencoder and generator architecture. Although we explore extended generator training, our pooling-and-decoder training budget remains fixed; whether longer first-stage training can improve performance at higher compression rates remains open. Finally, competitive generation quality relies heavily on guidance, as unguided performance noticeably degrades under compression.

AI use disclosure

We used generative AI tools to assist with manuscript drafting and editing, code and figure preparation. The authors reviewed and revised the AI-assisted material and take responsibility for the final text, methods, and reported results.

Ethics statement

PoolDINO reduces the computational cost of image generation, but does not address biases inherited from its training data or pretrained encoder. More efficient generation may also lower barriers to producing misleading or harmful imagery. These risks should be considered when deploying the method; our evaluation is limited to research benchmarks and does not establish its suitability for unrestricted use.

Reproducibility statement

An anonymized implementation is available at our anonymous repository. We will publicly release the code after review. Appendix A describes the model and training settings, and Appendices B–F document throughput measurement, classification and dense-prediction protocols, guidance sweeps, and operator analysis.

References

Ramón Calvo-González and François Fleuret. Laminating representation autoencoders for efficient diffusion, 2026. URL https://arxiv.org/abs/2602.04873.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging Properties in Self-Supervised Vision Transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9650–9660, October 2021.

Hao Chen, Yujin Han, Fangyi Chen, Xiang Li, Yidong Wang, Jindong Wang, Ze Wang, Zicheng Liu, Difan Zou, and Bhiksha Raj. Masked Autoencoders Are Effective Tokenizers for Diffusion Models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8145–8171. PMLR, 13–19 Jul 2025a. URL https://proceedings.mlr.press/v267/chen25v.html.

Hao Chen, Ze Wang, Xiang Li, Ximeng Sun, Fangyi Chen, Jiang Liu, Jindong Wang, Bhiksha Raj, Zicheng Liu, and Emad Barsoum. SoftVQ-VAE: Efficient 1-Dimensional Continuous Tokenizer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 28358–28370, June 2025b.

Yuan Gao, Chen Chen, Tianrong Chen, and Jiatao Gu. One Layer Is Enough: Adapting Pretrained Visual Encoders for Image Generation, 2025. URL http://arxiv.org/abs/2512.07829.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf.

Jonathan Ho and Tim Salimans. Classifier-Free Diffusion Guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021. URL https://openreview.net/forum?id=qw8AKxfYbI.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

Xingjian Leng, Jaskirat Singh, Yunzhong Hou, Zhenchang Xing, Saining Xie, and Liang Zheng. REPA-E: Unlocking VAE for End-to-End Tuning of Latent Diffusion Transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 18262–18272, October 2025.

Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 36115–36125, June 2026.

Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive Image Generation without Vector Quantization. In *Advances in Neural Information Processing Systems*, volume 37, pp. 56424–56445. Curran Associates, Inc., 2024. doi: 10.52202/079017-1797. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/66e226469f20625aaebddbe47f0ca997-Paper-Conference.pdf.

Yaron Lipman, Ricky T.Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow Matching for Generative Modeling. In *11th International Conference on Learning Representations, ICLR 2023*, May 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

William Peebles and Saining Xie. Scalable Diffusion Models with Transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4195–4205, October 2023.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115 (3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf.

Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *Computer Vision – ECCV 2012*, pp. 746–760. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-33715-4_54. URL https://doi.org/10.1007/978-3-642-33715-4_54.

Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seung Eun Yi, Michael Ramamonjisoa, Francisco Massa, Daniel Haziza, Luca Wehrstedt, Jianyuan Wang, Timothée Darcet, Théo Moutakanni, Leonel Sentana, Claire Roberts, Andrea Vedaldi, Jamie Tolan, John Brandt, Camille Couprie, Julien Mairal, Herve Jegou, Patrick Labatut, and Piotr Bojanowski. DINOv3. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=2NlGyqNjns. Featured Certification.

Jaskirat Singh, Xingjian Leng, Zongze Wu, Liang Zheng, Richard Zhang, Eli Shechtman, and Saining Xie. What matters for Representation Alignment: Global Information or Spatial Structure? In *International Conference on Learning Representations*, pp. 33807–33845, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/3929a7785bd56f57edcff0152ab41289-Paper-Conference.pdf.

Jaskirat Singh, Boyang Zheng, Zongze Wu, Richard Zhang, Eli Shechtman, and Saining Xie. Improved baselines with representation autoencoders, 2026b. URL https://arxiv.org/abs/2605.18324.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Jingfeng Yao, Bin Yang, and Xinggang Wang. Reconstruction vs. Generation: Taming Optimization Dilemma in Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15703–15712, June 2025.

Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation Alignment for Generation: Training Diffusion Transformers Is Easier Than You Think. In *International Conference on Learning Representations*, pp. 87400–87442, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/d9e42b4d7163931f3689d6d6fbaa11d0-Paper-Conference.pdf.

Boyang Zheng, Nanye Ma, Shengbang Tong, and Saining Xie. Diffusion Transformers with Representation Autoencoders. In *International Conference on Learning Representations*, pp. 35791–35820, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/3c4141c12660ad3625eb4ae845e0a6f9-Paper-Conference.pdf.

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene Parsing Through ADE20K Dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 633–641, July 2017. URL https://openaccess.thecvf.com/content_cvpr_2017/html/Zhou_Scene_Parsing_Through_CVPR_2017_paper.html.

Xingyu Zhou, Qifan Li, Xiaobin Hu, Hai Chen, and Shuhang Gu. Guiding a Diffusion Transformer with the Internal Dynamics of Itself. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11536–11545, June 2026.

A TRAINING AND MODEL DETAILS

We retain the two-stage RAE procedure. The first stage learns the local affine pooling operator and RGB decoder with the vision encoder frozen; the second freezes these components and trains the class-conditional generator. Figure 5 shows the full generator training graph. Tables 5–7 give the latent grids and model settings, and Table 8 reports component-wise forward costs.

RGB reconstruction objective. Following RAEv2, we train the pooling operator and image decoder with

$$\mathcal{L}_{\mathrm{RGB}} = \mathcal{L}_1 + \mathcal{L}_{\mathrm{LPIPS}} + 0.75\, a\, \mathcal{L}_{\mathrm{adv}}, \tag{8}$$

where $\mathcal{L}_1$ is mean absolute pixel error, $\mathcal{L}_{\mathrm{LPIPS}}$ is the perceptual reconstruction loss, and $\mathcal{L}_{\mathrm{adv}}$ is the negative mean discriminator score on reconstructed images. The adaptive weight a is the ratio of the gradient norms of $\mathcal{L}_1 + \mathcal{L}_{\mathrm{LPIPS}}$ and $\mathcal{L}_{\mathrm{adv}}$ at the decoder's final projection, with 10^{-6} added to the denominator and the ratio clipped to $[0, 10^4]$. Pixel and perceptual losses are active from the start; discriminator training begins after six epochs and the decoder's adversarial term after eight. During training, we add Gaussian noise to the pooled tokens before repetition, with a standard deviation sampled independently for each image from $\mathcal{U}(0, 0.8)$. Evaluation uses clean tokens and EMA parameters.

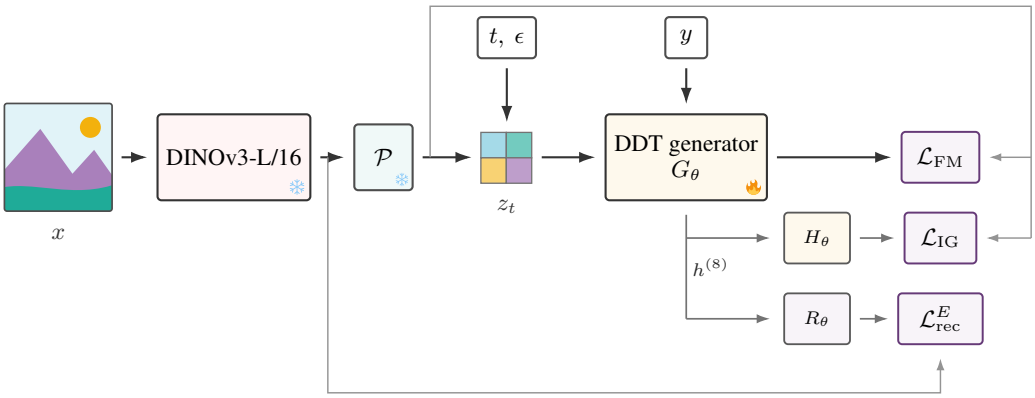

Figure 5: Training the pooled generator. The frozen encoder and pooling operator define the normalized clean latent $\bar{z}$. The generator predicts $\bar{z}_{\mathrm{full}}$ from $z_t = (1-t)\epsilon + t\bar{z}$, conditioned on time t and class y. From block-8 hidden states $h^{(8)}$, the internal-guidance head H_θ predicts the same clean pooled target, while the encoder-reconstruction head R_θ predicts the dense features z.

As illustrated in Figure 5, let R_θ denote the learned encoder-reconstruction head. At each pooled spatial position (i, j), it maps the hidden state $h^{(8)}_{i,j} \in \mathbb{R}^{d_h}$ to

$$R_\theta : \mathbb{R}^{d_h} \to \mathbb{R}^{p_y p_x D}, \qquad h^{(8)}_{i,j} \mapsto R_\theta\left(h^{(8)}_{i,j}\right). \tag{9}$$

This vector is reshaped into $p_y p_x$ separate D-dimensional predictions, one for every position in the original pooling window from which $\hat{z}_{i,j}$ was formed.

Table 5: Latent grids. Compression is relative to the 256-token control.

Pooling window	Latent grid	Tokens	Compression	κ
1×1 (control)	16×16	256	$1\times$	8
2×2	8×8	64	$4\times$	4
2×4	8×4	32	$8\times$	$\sqrt{8}$
4×2	4×8	32	$8\times$	$\sqrt{8}$
4×4	4×4	16	$16\times$	2

Table 6: Model architecture hyperparameters.

Hyperparameter	Image decoder	Flow encoder	Flow decoder
Transformer blocks	28	28	2
Hidden size	1,152	1,440	2,048
MLP width	4,096	3,840	5,461
Attention heads	16	20	16
Time tokens	–	4	–
Class tokens	–	8	–

Table 7: Optimization hyperparameters for both training stages.

Hyperparameter	Decoder Stage	Flow-Model Stage
Training epochs	16	80
Training steps	40,032	100,080
Global batch size	512	1,024
Optimizer	AdamW	Muon (AdamW fallback)
Base learning rate	2×10^{-4}	2×10^{-4}
Learning rate schedule	Cosine decay to 2×10^{-5}	Constant through epoch 25; linear decay to 2×10^{-5} at epoch 50
Warm-up epochs	1	–
Weight decay	0	0
Optimizer momentum	$(\beta_1 = 0.9, \beta_2 = 0.95)$	Muon 0.95, AdamW $(0.9, 0.95)$
Gradient clipping	None	1.0
EMA decay	0.9978	0.9995

Table 8: Estimated forward FLOPs per image, counting a multiply-add as two operations. Component columns report one evaluation. "Internal head" is the additional cost of returning the early clean-latent prediction. "Representation path" includes the dense-representation projection and pooling operation. Unguided generation counts 100 base generator evaluations and one image-decoder evaluation, excluding auxiliary heads and additional CFG evaluations.

Window	Tokens	Generator	Internal head	Repr. path	Decoder	Unguided generation
1×1	256	473.182G	2.887G	0.756G	224.216G	47.542T
2×2	64	128.653G	0.722G	1.292G	224.216G	13.090T
2×4	32	72.531G	0.361G	1.292G	224.216G	7.477T
4×4	16	44.612G	0.180G	1.292G	224.216G	4.685T

A.1 EXTENDED-TRAINING COMPUTE BUDGET

We compare the cost of extended generator training with the 80-epoch unpooled baseline. The per-update estimate includes frozen encoder and pooling operations, generator forward and backward passes, auxiliary losses, optimizer updates, and EMA updates. We count a multiply-add as two FLOPs and exclude first-stage decoder training and evaluation. With a global batch size of 1,024, each epoch contains 1,251 updates. The unpooled, 2×2, and 4×4 models require approximately 1,662.451, 599.874, and 340.648 TFLOPs per update, respectively. For reference, matching the baseline's total training cost would require

$$e_{2\times2}^{\mathrm{match}} = 80\,\frac{1662.451}{599.874} \simeq 221.71, \qquad e_{4\times4}^{\mathrm{match}} = 80\,\frac{1662.451}{340.648} \simeq 390.42 \tag{10}$$

epochs. Our completed 2×2 and 4×4 runs use 180 and 300 epochs, respectively, rather than these matching durations. Their estimated costs are 135.08 and 127.85 EFLOPs, compared with 166.38 EFLOPs for the unpooled baseline: approximately 81% and 77% of its training budget.

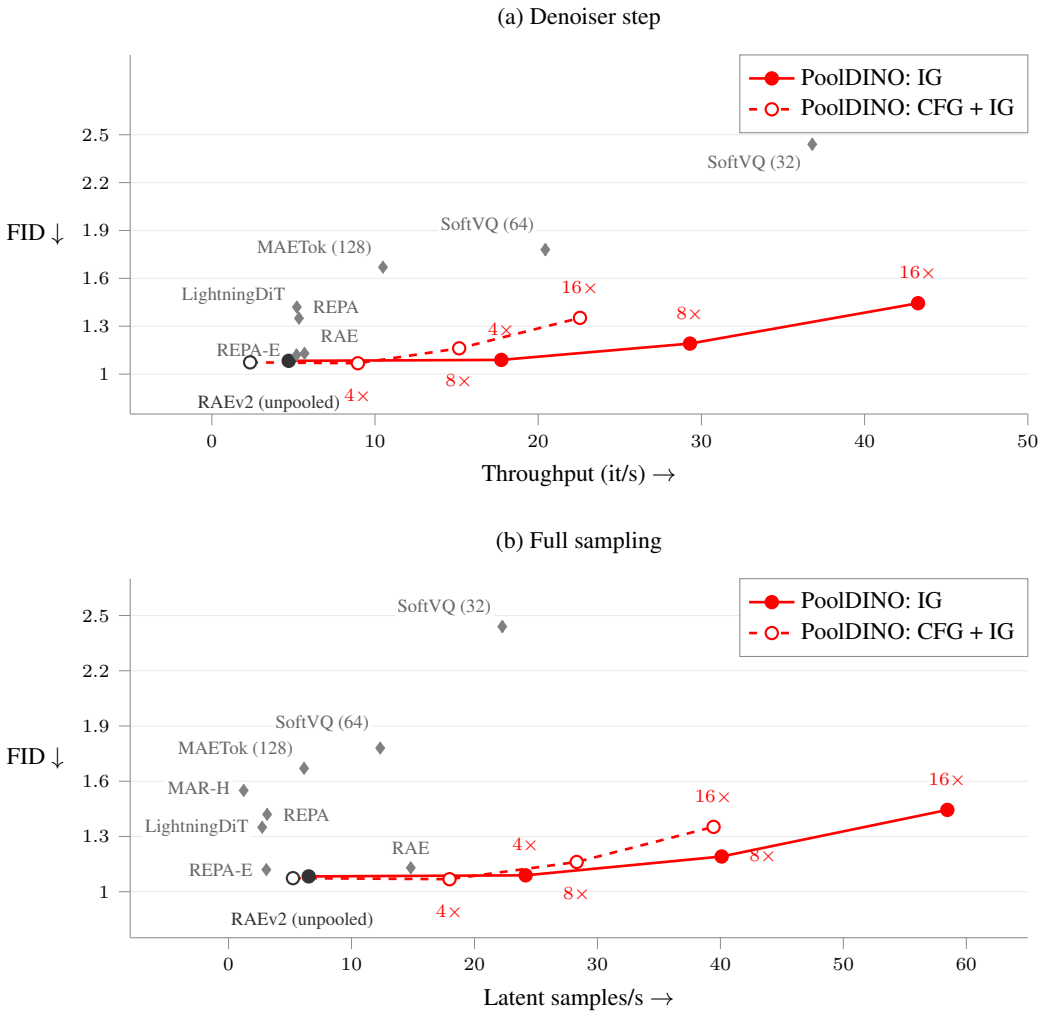

Figure 6: Quality–throughput comparison with both guidance settings on an H100 NVL at batch size 128. Our 80-epoch models use IG alone (solid, filled) or CFG+IG (dashed, hollow), with FID evaluated at 100 Euler steps. Point labels give token compression; black circles denote the unpooled RAEv2 reference. Literature baselines and timing conventions match Figure 1; full-sampling throughput excludes image decoding.

B THROUGHPUT MEASUREMENT

Figure 1 shows latent-sampling throughput for our models with IG alone. Figure 6 adds denoiser-step throughput and the CFG+IG results under the same measurement conventions.

Figure 6 reports the throughput of individual guided denoiser steps (top) and complete latent-sampling trajectories (bottom). Our models follow the architecture detailed in Table 6, varying only the spatial input grid. For all measurements, we execute model computation in BF16 and return full-precision model outputs.

Means and sample standard deviations are computed across 30 runs of 20 denoiser steps or one complete sampling trajectory, measured after compilation and a warmup phase. PoolDINO trajectory timings use CUDA events with device synchronization after at least five warmup trajectories and five seconds of warmup. Latent samples/s in the bottom panel counts completed latent samples, not end-to-end RGB outputs.

The throughput for our models is measured under two guidance settings. *IG alone* uses a conditional model evaluation and its block-8 internal head. *CFG+IG* adds an unconditional evaluation when CFG is active.

For the 100-step benchmarks, full trajectories use the same pooling-dependent time shifts as the quality evaluations: 8, 4, $\sqrt{8}$, and 2 for 1×1, 2×2, 2×4, and 4×4, respectively. IG is active on $[0, 0.9]$ and CFG on $(0.3, 1)$ in noise-to-data time.

For context, Figure 1 includes the original guided XL implementations of RAE (Zheng et al., 2026), LightningDiT (Yao et al., 2025), E2E-VAE + REPA from REPA-E (Leng et al., 2025), MAETok (Chen et al., 2025a), SoftVQ-VAE (Chen et al., 2025b), and REPA (Yu et al., 2025). MAR-H (Li et al., 2024) appears only in the full-sampling panel, since its autoregressive iterations are not comparable denoiser steps. We use MAETok-B-128, SoftVQ-B-64, and SoftVQ-L-32 with SiT-XL, and the released 4M-iteration SiT-XL/2 for REPA. RAE employs a separate small autoguidance model; the other systems use their released CFG recipes. Literature FIDs are published results, not re-evaluations under benchmark precision, and training budgets, sampling recipes, and class-balance protocols differ. The REPA-E point is the 800-epoch E2E-VAE + REPA system, not the 80-epoch jointly trained variant.

RAE and LightningDiT use shifted Euler sampling with 49 and 249 denoiser evaluations, respectively. REPA-E, MAETok, SoftVQ-VAE, and REPA use 250-evaluation Euler–Maruyama sampling. MAR-H uses 256 autoregressive iterations, each with 100 DDPM steps; its internal Transformer decoder and diffusion-loss network are included. Full trajectories respect each method's guidance windows. These measured sampling rates are not obtained by dividing isolated step throughput by the number of evaluations.

The literature baselines use PyTorch/Inductor, while our models use JAX/XLA, so the comparison includes software-stack differences. All use cuDNN attention. The PoolDINO trajectory benchmark uses cuBLAS.

Figure 7 gives the comparison using Inception Score, retaining the 100-step settings for our models.

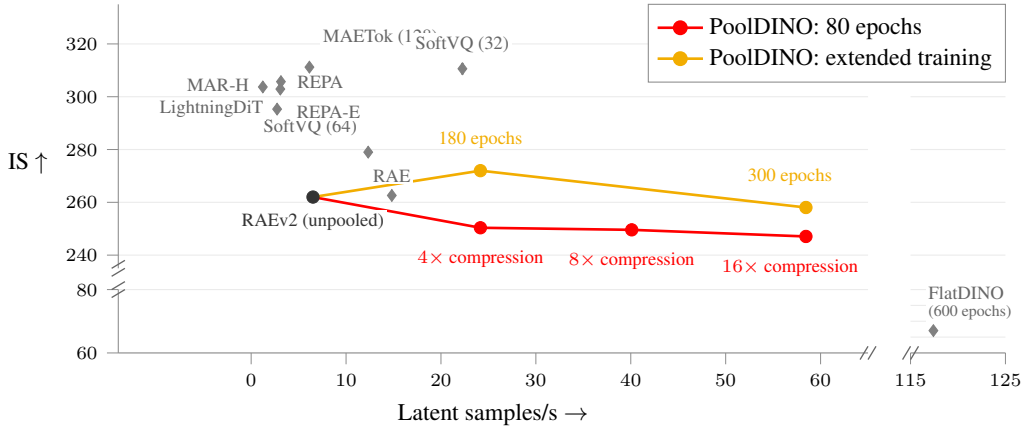

Figure 7: Inception Score versus latent-sampling throughput on an H100 NVL at batch size 128. PoolDINO and the unpooled RAEv2 reference use IG alone at 100 Euler steps, including the extended-training checkpoints. IS is taken at the FID-selected guidance settings, not maximized separately.

B.1 Sampling-step ablation

We compare 50 and 100 Euler steps while keeping each checkpoint's selected IG scale fixed. Across all evaluated pooling geometries and training durations, halving the sampling steps doubles latent-sampling throughput with little change in generation quality (Table 9). The absolute FID difference remains below 0.04 in every configuration.

For the 300-epoch 4×4 generator, we further vary the step count from 10 to 100 (Figure 9). Figure 8 shows all five measured 50-step PoolDINO configurations alongside the literature baselines, including

Table 9: IG-only generation with 100 versus 50 Euler steps. Throughput is latent samples/s on an H100 NVL at batch size 128; ± reports timing standard deviation.

			FID ↓		Latent samples/s ↑	
Pooling	Epochs	s_{IG}	100 steps	50 steps	100 steps	50 steps
2×2	80	1.75	1.0890	1.1164	24.16 ± 0.05	48.44 ± 0.17
2×2	180	2.00	1.0525	1.0629	24.16 ± 0.05	48.44 ± 0.17
2×4	80	2.00	1.1913	1.2283	40.10 ± 0.07	80.42 ± 0.33
4×4	80	2.75	1.4442	1.4429	58.46 ± 0.20	117.13 ± 0.38
4×4	300	2.75	1.2882	1.2981	58.46 ± 0.20	117.13 ± 0.38

FlatDINO. Figure 1 and the main generation tables retain 100 steps for our models to isolate the effect of training duration.

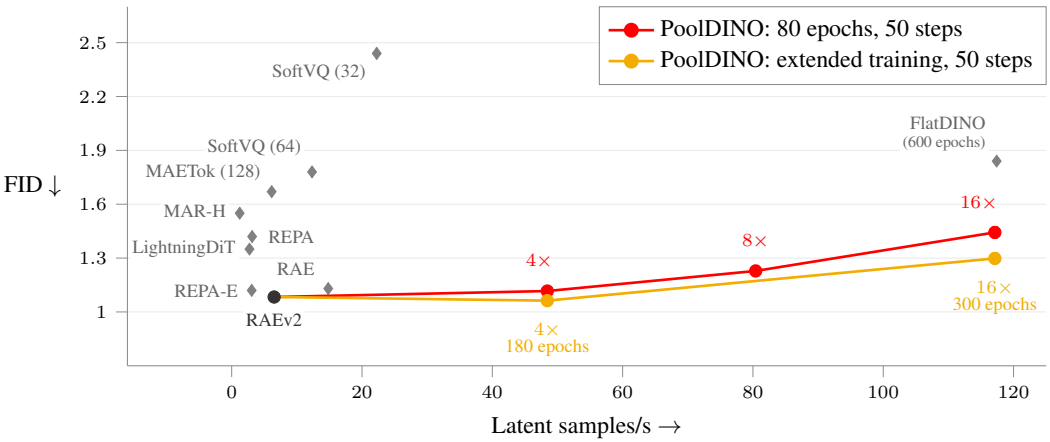

Figure 8: Generation quality versus latent-sampling throughput with 50-step sampling for PoolDINO, using IG alone on an H100 NVL at batch size 128. Red points show the 80-epoch models; gold points show extended training. The RAEv2 reference uses IG with 100 steps. Literature points, including FlatDINO, retain their published FIDs and measured throughput under their respective sampling protocols.

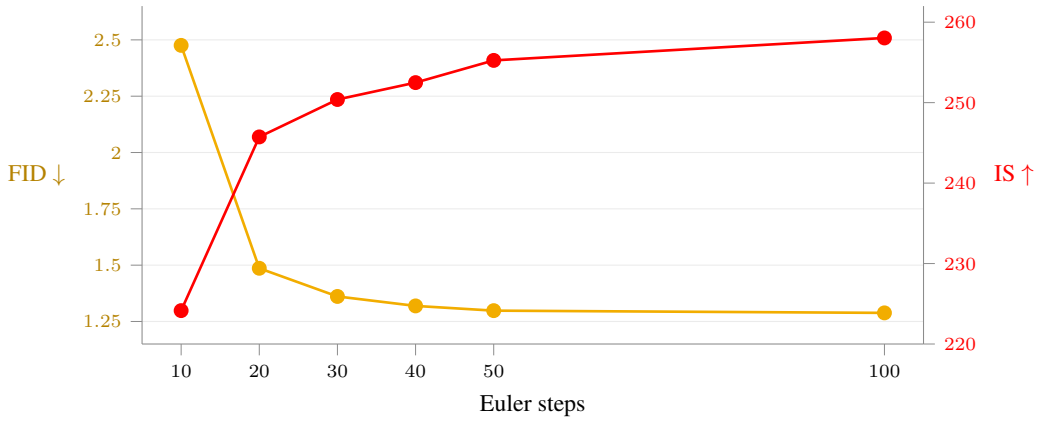

Figure 9: Sampling-step ablation for the 4×4 generator trained for 300 epochs. Gold shows FID (left axis); red shows IS (right axis). The IG scale is fixed at 2.75 rather than retuned for each step count.

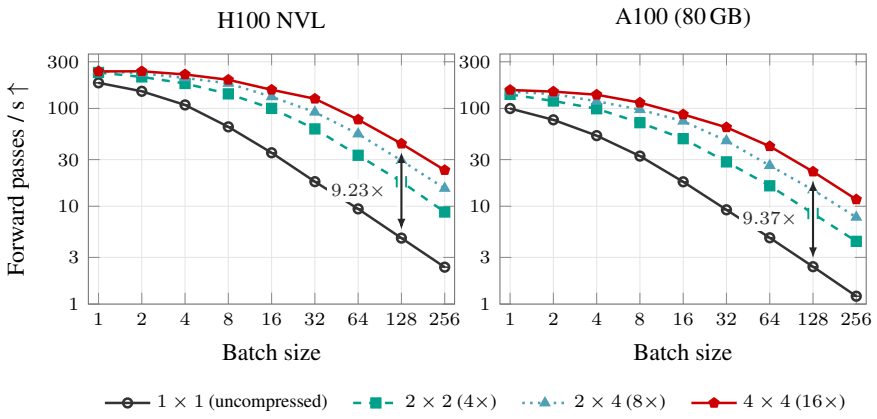

Figure 10: Denoiser forward-pass speed on a single GPU, using the flow-model architecture in Table 6. Legend entries give pooling windows and token compression. Each point is the mean of 30 measurements averaging 20 forward calls; error bars show one standard deviation and are mostly smaller than the markers. Benchmarks use BF16 computation and include host dispatch and synchronization, but not end-to-end image generation.

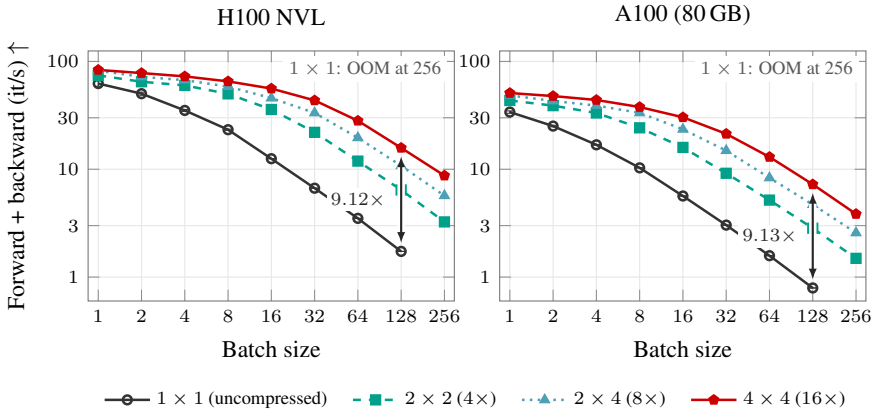

Figure 11: Generator forward–backward throughput on a single GPU. Points show mean iterations/s across 30 measurements of 20 iterations. Timing includes the forward pass, main-prediction MSE loss, backward pass, and dispatch/synchronization; it excludes optimizer updates, auxiliary losses, and the encoder/decoder. The uncompressed model runs out of memory at batch size 256 on both GPUs.

B.2 Batch-size and hardware sweeps

Figures 10 and 11 compare the same generator architecture across pooling grids on H100 NVL and A100 GPUs. The forward–backward benchmark measures the base generator and its main prediction loss; it excludes optimizer updates and auxiliary objectives.

Table 10: ImageNet-1K k-NN and linear-probe top-1 accuracy (%). The uncompressed RAEv2 representation z is included as a reference. Best and second-best methods at each compression rate are bold and underlined.

	k-NN				*Linear probe*			
Window	Learned	Average	PCA	Random	Learned	Average	PCA	Random
Uncompressed z	76.03				85.31			
2×2	72.16	<u>76.00</u>	**76.67**	75.78 ± 0.06	83.47	**85.33**	84.35	<u>84.63 ± 0.06</u>
2×4	68.50	<u>76.00</u>	**76.58**	75.77 ± 0.03	82.35	**85.33**	84.10	<u>84.50 ± 0.04</u>
4×2	68.17	<u>76.00</u>	**76.59**	75.68 ± 0.05	82.35	**85.34**	84.12	<u>84.43 ± 0.07</u>
4×4	63.22	<u>76.00</u>	**76.39**	75.68 ± 0.04	80.80	**85.33**	83.77	<u>84.21 ± 0.04</u>

C IMAGE CLASSIFICATION DETAILS

We freeze the encoder and pooling operator. For both ImageNet-1K splits, we apply deterministic resizing and center cropping to 256×256, followed by ImageNet normalization, without random augmentation. We spatially average each pooled field $\hat{z}$ into a 1,024-dimensional vector and normalize it to unit ℓ_2 norm. The uncompressed reference uses the spatial mean of z. All pooling methods use the same encoder features and classifier input dimension.

k-NN. Following the weighted voting procedure of DINO (Caron et al., 2021), we retrieve the k training features with highest cosine similarity to each validation feature. Each neighbor votes for its class with weight $\exp(s/\tau)$, where s is its cosine similarity and $\tau = 0.07$. The class with the largest total weight is predicted. We report the best validation top-1 accuracy over $k \in \{10, 20, 100\}$.

Linear probe. We train a single affine classifier with cross-entropy on the cached training features, keeping the representation fixed. We use LARS for 90 epoch-equivalents with batch size 16,384, zero weight decay, and gradient-norm clipping at 10. Each update samples a batch of distinct training examples. The learning rate increases linearly from zero during the first 10% of updates and then decays to zero with a cosine schedule. We sweep peak learning rates $\{1.6, 3.2, 6.4, 12.8\}$ and report the highest validation top-1 accuracy among the final classifiers.

Pooling baselines. Table 10 compares learned pooling with average pooling, training-fitted local PCA, and random projection. For a window of m patches, PCA centers the flattened mD-dimensional features with the training mean and projects onto the leading $D = 1024$ components, without whitening. The components are fitted by randomized PCA on training blocks and held fixed for both classification splits. Random projections use the same training mean: we draw a standard Gaussian matrix in $\mathbb{R}^{mD \times D}$, orthonormalize its columns by QR decomposition, and use its transpose as the projection. Each projection is shared across all spatial windows. We report the mean and standard deviation across projection draws, selecting the best probe setting separately for each draw. These probes evaluate classification from spatially averaged features; the dense-task evaluations retain the token grid.

D DENSE PREDICTION DETAILS

We report complete validation metrics for task models trained from scratch and for models initialized from the corresponding RGB decoder. The tables select the checkpoint with the best primary validation metric. The scratch controls separate information retained by the representation from transfer provided by RGB-decoder initialization. For C semantic classes and N valid depth pixels, the primary metrics are

$$\mathrm{mIoU} = \frac{1}{C}\sum_{c=1}^{C}\frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c + \mathrm{FN}_c}, \qquad \mathrm{AbsRel} = \frac{1}{N}\sum_{j=1}^{N}\frac{|\hat{d}_j - d_j|}{d_j}, \tag{11}$$

where d_j and $\hat{d}_j$ are the ground-truth and predicted depths. The threshold accuracy δ_i is the fraction of pixels satisfying

$$\max\left(\frac{\hat{d}_j}{d_j}, \frac{d_j}{\hat{d}_j}\right) < 1.25^i, \qquad i \in \{1, 2, 3\}. \tag{12}$$

Shared training setup. We freeze the encoder and EMA pooling operator and repeat the pooled tokens to the original 16×16 grid. A ViT-XL task decoder with a newly initialized output head predicts segmentation logits or log depth. RGB initialization copies the corresponding image decoder's Transformer parameters, not its RGB output head. Both initialization settings use AdamW with batch size 32, $(\beta_1, \beta_2) = (0.9, 0.999)$, weight decay 0.05, and gradient-norm clipping at 3. The learning rate warms up linearly from zero to 2×10^{-4} over two epochs and then follows a cosine decay to 10^{-6}. We train for 80 epochs on ADE20K and 50 on NYUv2. Images use ImageNet normalization and are resized to 256×256 for the frozen encoder.

ADE20K protocol. We use the official 150-class training and validation splits. Training applies random scaling in $[0.5, 2]$, 512×512 crops, horizontal flips, and color jitter. Crops are retried up to ten times to avoid a single class occupying more than 75% of valid pixels. Validation preserves aspect ratio, resizes the longest side to 512 pixels, and pads to a square. The decoder's logits are bilinearly resized to the 512×512 target grid. Training uses pixelwise cross-entropy, excluding unlabeled and padded pixels; these pixels are also excluded from evaluation. We select the checkpoint with the highest validation mIoU.

NYUv2 protocol. We retain 480×640 depth targets in meters and augment training images with paired horizontal flips and color jitter. The decoder predicts log depth, which is converted to metric depth before bilinear resizing to the target resolution. On valid training pixels, the loss is $\sqrt{\langle e^2 \rangle - 0.5\langle e \rangle^2 + 10^{-6}}$, where $e = \log \hat{d} - \log d$ and brackets denote the mean over valid pixels; no gradient-matching term is used. Valid depths lie in $[0.1, 10]$ meters. Evaluation additionally restricts pixels to rows $[45, 471)$ and columns $[41, 601)$ in zero-based coordinates, clips predictions to the same depth range, and uses no scale alignment. Metrics aggregate over valid pixels, and we select the checkpoint with the lowest validation AbsRel.

Training from scratch. We initialize the ViT-XL task model randomly while keeping the frozen representations and task-specific training recipes fixed.

Table 11: ADE20K semantic segmentation with task models trained from scratch. Accuracy metrics are percentages.

Spatial pooling	Best epoch	Val. loss ↓	mIoU ↑	Mean acc. ↑	Pixel acc. ↑
Uncompressed	75	0.671	46.860	58.060	81.260
Learned 2×2	75	0.807	39.380	50.100	77.480
Average 2×2	65	0.735	42.591	53.477	78.905
Learned 4×4	80	0.863	38.610	48.850	77.780
Average 4×4	75	0.866	36.758	46.969	75.715

Table 12: NYUv2 monocular depth estimation with task models trained from scratch. The δ metrics are percentages.

Spatial pooling	Best epoch	AbsRel ↓	RMSE ↓	Log RMSE ↓	SiLog ↓	δ_1 ↑	δ_2 ↑	δ_3 ↑
Uncompressed	12	0.082	0.409	0.122	0.122	94.170	99.030	99.780
Learned 2×2	21	0.090	0.455	0.134	0.134	92.780	98.700	99.720
Average 2×2	14	0.087	0.434	0.131	0.131	93.200	98.720	99.730
Learned 4×4	27	0.102	0.490	0.148	0.148	90.440	98.260	99.610
Average 4×4	15	0.100	0.486	0.149	0.149	90.580	98.060	99.560

Table 13: ADE20K semantic segmentation with task models initialized from the corresponding RGB decoder. Accuracy metrics are percentages.

Spatial pooling	Tokens	Compression	Best epoch	Val. loss ↓	mIoU ↑	Mean acc. ↑	Pixel acc. ↑
Uncompressed	256	1×	80	0.694	47.810	58.850	81.770
Learned 2×2	64	4×	60	0.709	45.380	56.420	81.010
Learned 4×4	16	16×	80	0.788	42.500	53.240	79.960

RGB-decoder initialization. For the learned tokenizers, we initialize the ViT-XL task model from the corresponding RGB decoder and otherwise retain the same downstream protocol.

Table 14: NYUv2 monocular depth estimation with task models initialized from the corresponding RGB decoder. The δ metrics are percentages.

Spatial pooling	Best epoch	AbsRel ↓	RMSE ↓	Log RMSE ↓	SiLog ↓	δ_1 ↑	δ_2 ↑	δ_3 ↑
Uncompressed	14	0.081	0.411	0.123	0.123	94.330	98.980	99.750
Learned 2×2	27	0.089	0.456	0.133	0.133	92.800	98.720	99.700
Learned 4×4	24	0.101	0.495	0.147	0.147	90.610	98.220	99.610

E GUIDANCE ABLATIONS

We compare CFG, internal guidance, and guidance from the encoder-reconstruction head. All evaluations use 50,000 ImageNet-256 samples and 100 Euler steps. The sweeps below use the 80-epoch models (Table 18); the longer-trained 2×2 and 4×4 checkpoints are included separately in the summary tables (Tables 15 and 16).

E.1 FULL GENERATION COMPARISON

Table 15: ImageNet-256 generation at 100 Euler steps. Models train for 80 epochs, except the extended-training 2×2 (180 epochs) and 4×4 (300 epochs) variants. Guided columns report the lowest-FID settings from our guidance scale sweeps (the corresponding scales are in Table 16); – denotes an unevaluated setting. Best and second-best metrics in each column are bold and underlined.

	Unguided		CFG		IG		CFG + IG	
Variant	FID ↓	IS ↑	FID ↓	IS ↑	FID ↓	IS ↑	FID ↓	IS ↑
1×1	**1.53**	**226.63**	**1.45**	237.80	<u>1.08</u>	<u>262.00</u>	1.07	267.15
2×2	3.00	184.49	<u>1.99</u>	228.10	1.09	250.36	<u>1.07</u>	268.50
2×4	4.91	161.42	2.33	<u>238.80</u>	1.19	249.58	1.16	273.06
4×2	4.97	159.67	2.41	237.63	1.21	249.96	1.19	<u>273.98</u>
4×4	8.16	132.67	2.87	**249.72**	1.44	247.07	1.35	**282.29**
2×2 (180 epochs)	<u>2.78</u>	<u>192.37</u>	–	–	**1.05**	**272.00**	**1.05**	272.80
4×4 (300 epochs)	7.19	146.13	–	–	1.29	258.04	–	–

For the 80-epoch models, while IG alone improves generation across all compression rates, combining it with CFG yields the lowest overall FID, with the largest additional reduction occurring at $16\times$ compression. However, this combined advantage appears to diminish with longer training: for the 180-epoch 2×2 checkpoint, adding CFG to IG yields no further FID improvement.

Table 16: Guidance scales selected for Tables 2, 3, and 15. For CFG alone, $s_{\mathrm{IG}} = 1$; for IG alone, $s_{\mathrm{CFG}} = 1$. Both scales are one for unguided sampling.

	CFG	IG	CFG + IG	
Variant	s_{CFG}	s_{IG}	s_{CFG}	s_{IG}
1×1	3.00	1.75	2.50	1.78
2×2	3.00	1.75	1.75	1.75
2×4	3.00	2.00	1.50	2.00
4×2	3.00	2.00	1.50	2.00
4×4	3.00	2.75	1.75	2.25
Average 2×2	–	2.00	–	–
2×2 (180 epochs)	–	2.00	2.00	2.00
4×4 (300 epochs)	–	2.75	–	–

E.2 ISOLATED GUIDANCE SWEEPS

DINO patch guidance. The encoder-reconstruction head provides an alternative guidance signal. We first pass its predicted dense encoder field through the frozen tokenizer and the same latent normalization:

$$\bar{z}^{E}_{\mathrm{rec}} = \frac{\mathcal{P}\big(R_\theta(h^{(8)})\big) - \mu_{\hat{z}}}{\sigma_{\hat{z}}}. \tag{13}$$

The guided prediction is

$$\bar{z}^{E}_{\mathrm{guided}} = \bar{z}_{\mathrm{full}} + s^{E}_{\mathrm{rec}} \left(\bar{z}_{\mathrm{full}} - \bar{z}^{E}_{\mathrm{rec}}\right), \tag{14}$$

where $s^{E}_{\mathrm{rec}} = 0$ is neutral. We refer to this procedure as DINO patch guidance.

We test CFG, DINO patch guidance, and internal guidance in isolation. In each sweep, we vary one guidance scale and disable the other two methods, using EMA weights and the same validation conditions. CFG and internal guidance are neutral at scale one, whereas DINO patch guidance is neutral at zero.

Table 17: Sampling settings for the guidance ablations. Intervals use noise-to-data time: $t = 0$ is noise and $t = 1$ is clean data.

Method	Euler steps	Guidance scale	Guidance interval	Neutral value
Unguided	100	–	–	CFG 1
Internal guidance	100	Swept	$[0, 0.9]$	1
DINO patch guidance	100	Swept	$[0, 1]$	0

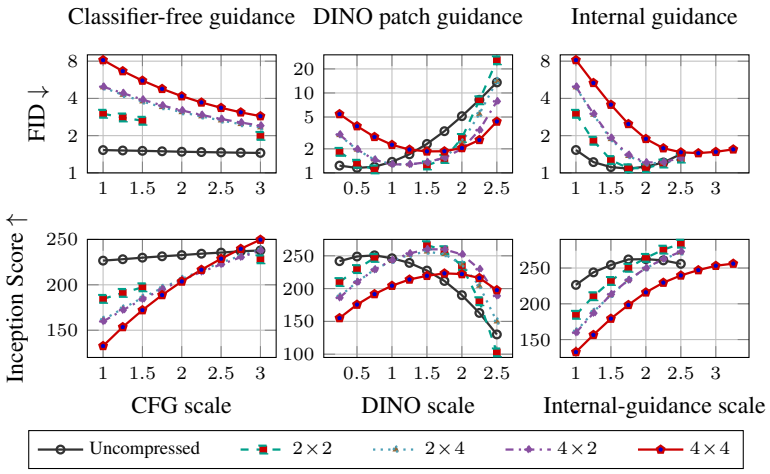

Figure 12: Isolated guidance-scale sweeps on ImageNet-256. Top: FID on a logarithmic scale. Bottom: Inception Score. Missing points in the corrected 2×2 sweeps are left blank.

Internal guidance gives the lowest observed isolated FID at every compression rate. CFG alone improves throughout the tested range, but becomes less effective as compression increases. DINO patch guidance has a narrower optimum and degrades rapidly at large scales. Inception Score often continues to improve after FID reaches its minimum.

E.3 Joint guidance searches

We jointly vary CFG with each auxiliary guidance method. The search uses irregular refinement grids, so the figures show measured configurations without interpolating unevaluated settings.

Table 18: Metrics at the lowest-FID point in each joint guidance sweep for the 80-epoch models. Both auxiliary methods are combined with CFG.

	DINO patch guidance		Internal guidance	
Variant	FID ↓	IS ↑	FID ↓	IS ↑
2×2	**1.06**	272.83	1.07	268.50
2×4	1.17	279.68	**1.16**	273.06
4×2	1.20	280.14	**1.19**	273.98
4×4	**1.35**	295.39	**1.35**	282.29

After joint tuning with CFG, the two auxiliary guidance methods achieve similar FID. DINO patch guidance gives the lowest observed value at 2×2, while internal guidance is stronger at 2×4 and 4×2; their 4×4 results are nearly identical. These comparisons concern sampling-time guidance: both auxiliary heads were trained in every model.

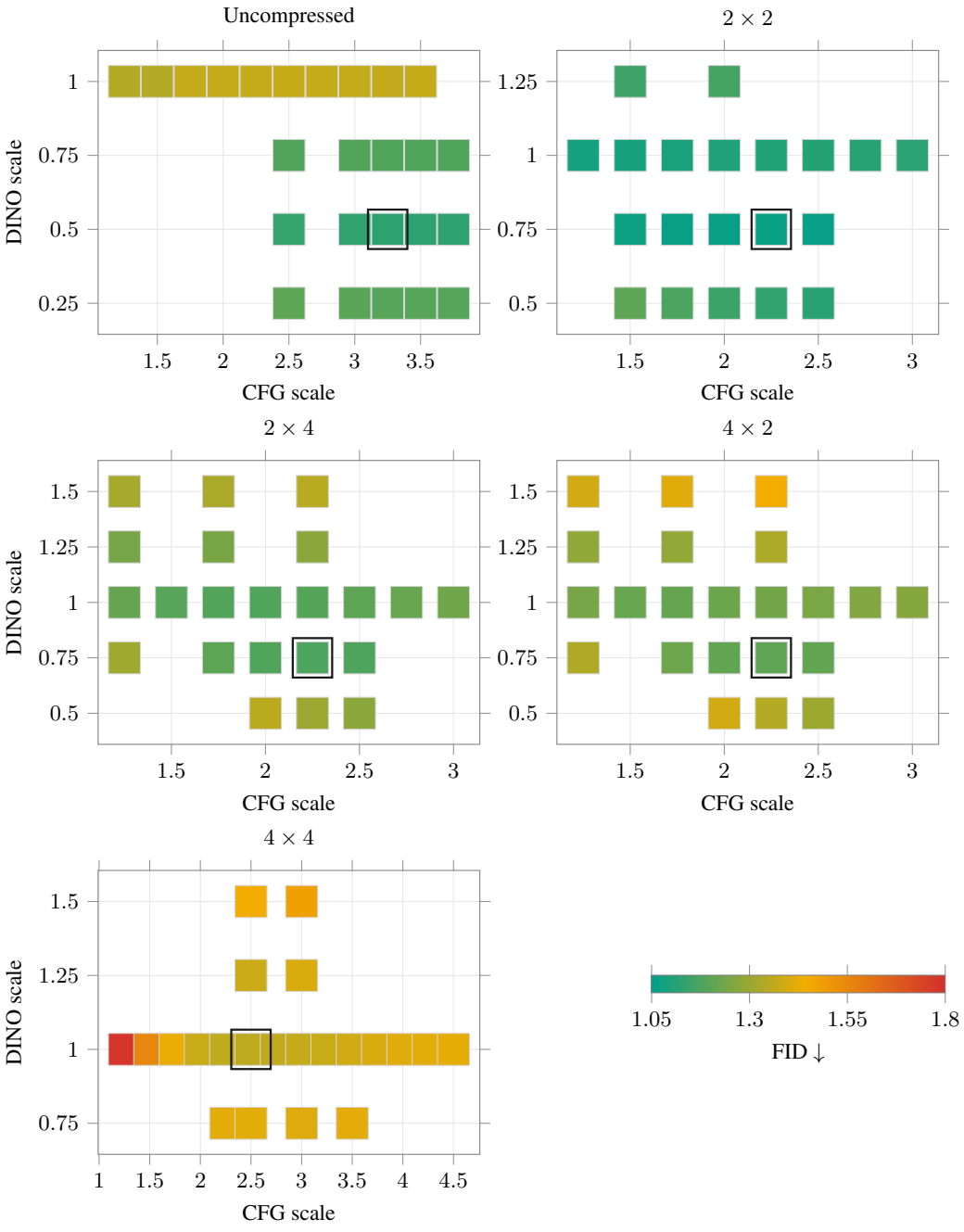

Figure 13: Joint CFG and DINO patch guidance search. Squares show evaluated 100-step configurations; the black outline marks the lowest FID in each panel.

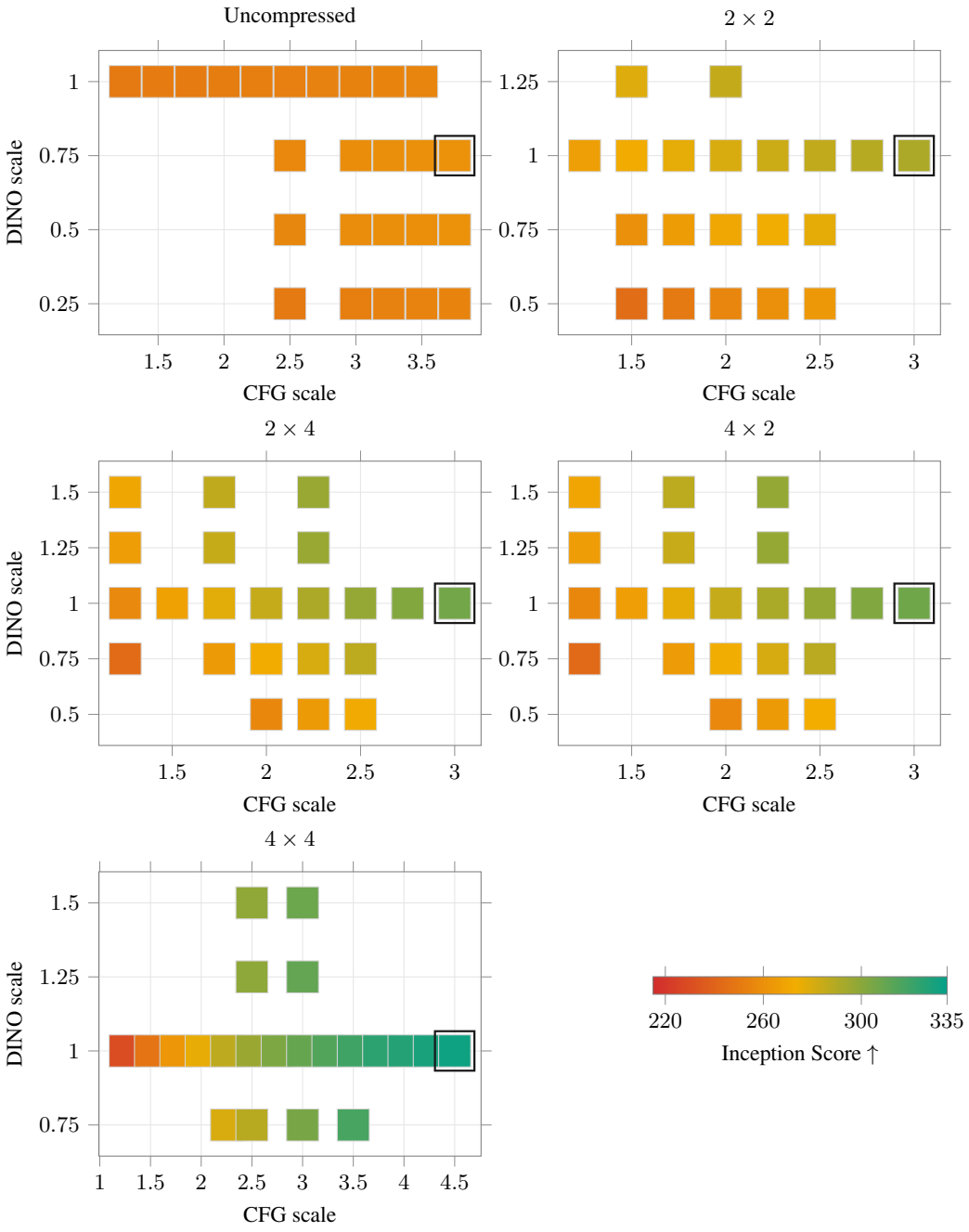

Figure 14: Inception Score for the joint CFG and DINO patch guidance search. The black outline marks the highest score in each panel.

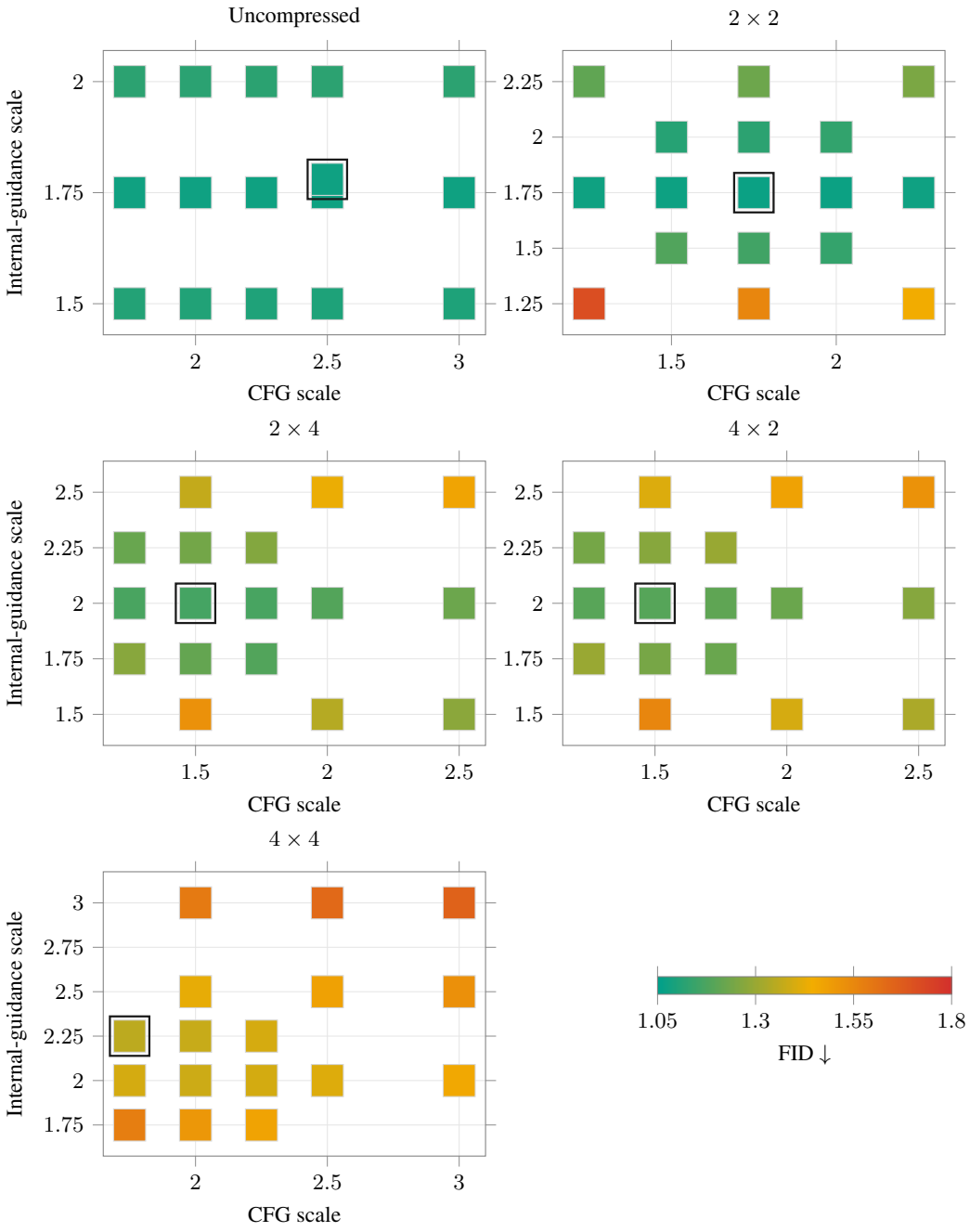

Figure 15: Joint CFG and internal-guidance search. Squares show evaluated 100-step configurations; the black outline marks the lowest FID in each panel.

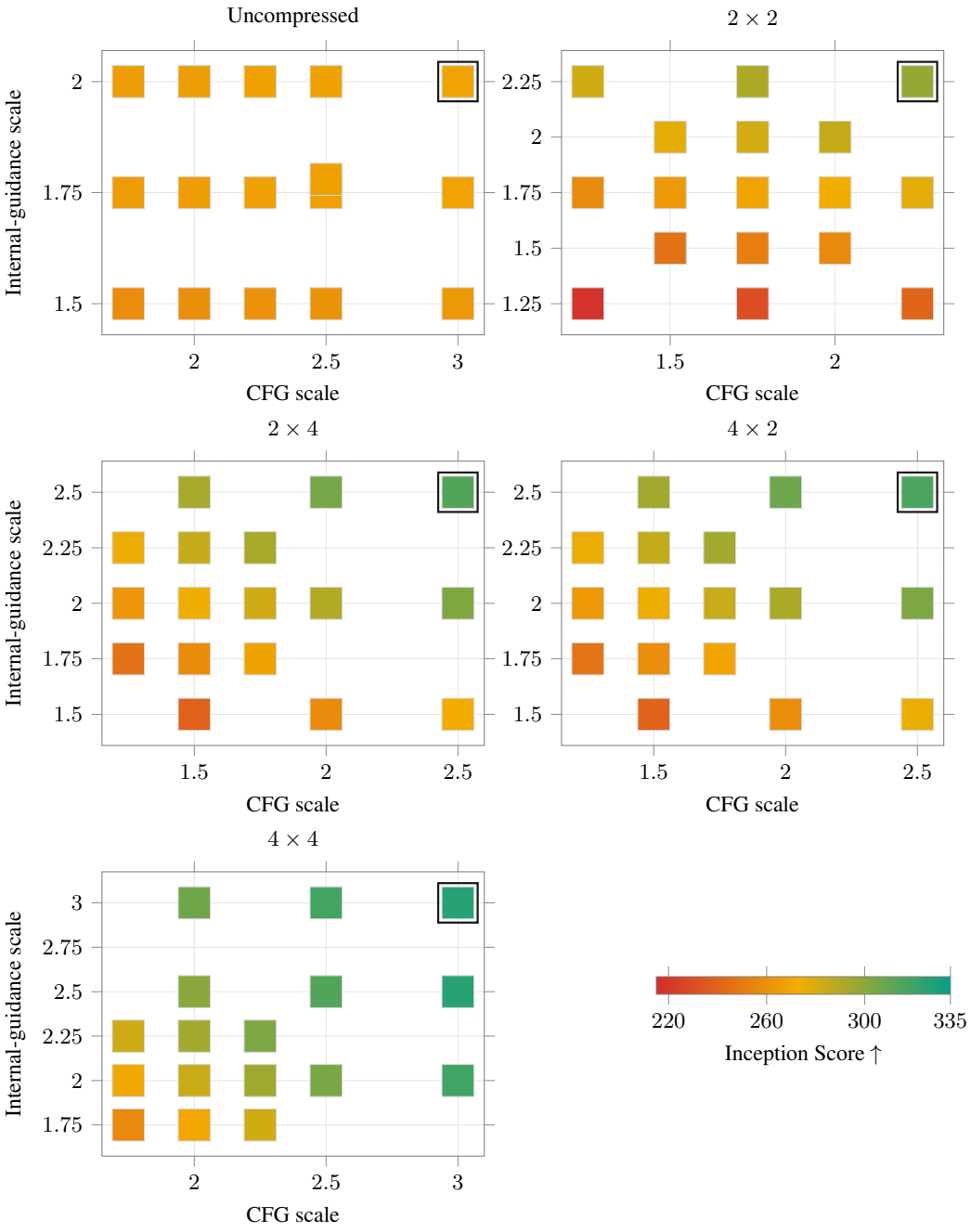

Figure 16: Inception Score for the joint CFG and internal-guidance search. The black outline marks the highest score in each panel.

Table 19: Learned pooling row-space analysis. Alignment scores are adjusted so that zero indicates the overlap expected between random subspaces of the same dimension and one indicates identical subspaces. Energy captured is the squared validation feature energy preserved by the learned pooling operator's row space (after centering with the training mean), expressed as a percentage of that captured by PCA with the same output dimension.

Window	Average alignment	PCA alignment	Energy captured (% of PCA)
2×2	0.5000	0.2798	67.60
2×4	0.3530	0.2114	52.16
4×2	0.3501	0.2082	52.05
4×4	0.2268	0.1524	38.16

F LEARNED POOLING OPERATOR ANALYSIS

Table 19 shows that learned pooling selects a row space distinct from spatial averaging and local PCA. Its adjusted alignment with both decreases as compression increases, and it captures less held-out feature variation than the training-fitted PCA baseline. These measurements characterize the learned operator but do not establish which retained directions support reconstruction or generation. The computation is described below.

Let $x \in \mathbb{R}^{mC}$ be a flattened pooling window containing $m = p_x p_y$ patches with $C = 1024$ channels. Learned pooling applies

$$y = Ax + b, \qquad A \in \mathbb{R}^{C \times mC}. \tag{15}$$

The input directions retained by the pooled token form the row space of A; the bias does not affect this space. We represent it with an orthonormal basis $Q_A \in \mathbb{R}^{mC \times C}$ obtained from the right singular vectors of A. All analyzed operators have numerical rank C.

Average pooling retains the spatially constant directions, with basis

$$Q_{\text{avg}} = \frac{1}{\sqrt{m}} \begin{bmatrix} I_C & I_C & \cdots & I_C \end{bmatrix}^\top. \tag{16}$$

For PCA, Q_{PCA} contains the leading C components fitted on centered training blocks using a randomized approximation. Because it is fitted on training data, it is not guaranteed to strictly maximize variance on the validation set. We measure alignment with either reference basis Q_B as

$$O(A, B) = \frac{1}{C} \left\| Q_A^\top Q_B \right\|_F^2 = \frac{1}{C} \sum_{i=1}^{C} \cos^2 \theta_i, \tag{17}$$

where θ_i are their principal angles. The score ranges from zero for orthogonal subspaces to one for identical subspaces.

Rate-matched random subspaces have expected overlap $1/m$. This expectation holds for independent, uniformly oriented C-dimensional subspaces of $\mathbb{R}^{mC}$. To compare window sizes, we report the adjusted score

$$O_{\text{adj}} = \frac{O - 1/m}{1 - 1/m}. \tag{18}$$

This maps the random expectation to zero and identical subspaces to one. The adjusted score provides a normalized geometric comparison across pooling dimensions, not a statistical significance test, and can take negative values. Table 19 reports adjusted alignment, while Table 20 includes both raw and adjusted values.

The average-pooling subspace is also the zero-frequency, or DC, component of a two-dimensional discrete cosine transform. Thus, $O(A, \text{avg})$ measures alignment with the spatial average. The complement $1 - O(A, \text{avg})$ measures the fraction of orthonormalized row-space energy outside the spatial-average subspace, not the fraction of input variance retained.

The analysis uses the clean EMA tokenizer at decoder step 40032. For each window geometry, we sample 4096 blocks from one ImageNet training image per class and 4096 blocks from one disjoint validation image per class. A randomized C-component PCA is fitted only on the training blocks.

Table 20: Complete learned pooling row-space results. "Random" is the expected raw overlap of random subspaces of the same dimension. Energy captured is reported as a percentage of rate-matched PCA.

		Average overlap		PCA overlap		
Window	Random	Raw	Adjusted	Raw	Adjusted	Energy (% PCA)
2×2	0.2500	0.6250	0.5000	0.4598	0.2798	67.60
2×4	0.1250	0.4339	0.3530	0.3100	0.2114	52.16
4×2	0.1250	0.4313	0.3501	0.3072	0.2082	52.05
4×4	0.0625	0.2751	0.2268	0.2054	0.1524	38.16

The analysis uses one checkpoint per geometry and a finite sample of blocks; small differences should therefore be interpreted cautiously.

After centering validation blocks X with the training mean, we measure the fraction of squared feature energy captured by an orthonormal basis Q:

$$V(Q) = \frac{\|XQ\|_F^2}{\|X\|_F^2}. \tag{19}$$

We report $100\, V(Q_A)/V(Q_{\mathrm{PCA}})$, comparing learned pooling with the rate-matched PCA baseline. This projection-based quantity is invariant to kernel scaling. Raw-kernel DCT energy is not invariant to invertible output-channel mixing and is therefore excluded as an intrinsic retained-subspace statistic.

G GENERATED SAMPLES

We show 48 randomly selected samples per model in eight-column, six-row grids. Corresponding positions have the same class condition across all grids. Generation uses 100 Euler steps and IG alone at each model's selected scale.

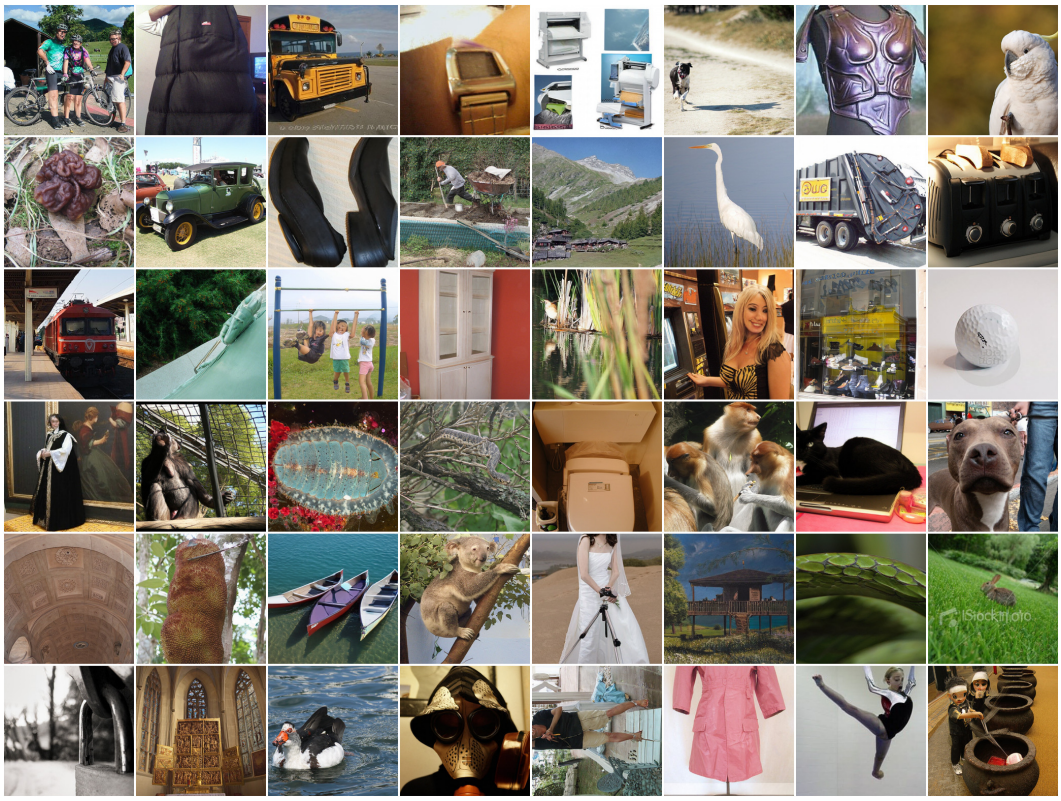

Figure 17: Unpooled RAEv2 reference (1×1), trained for 80 epochs, with $s_{\mathrm{IG}} = 1.75$ and no CFG. Samples are randomly selected without quality filtering.

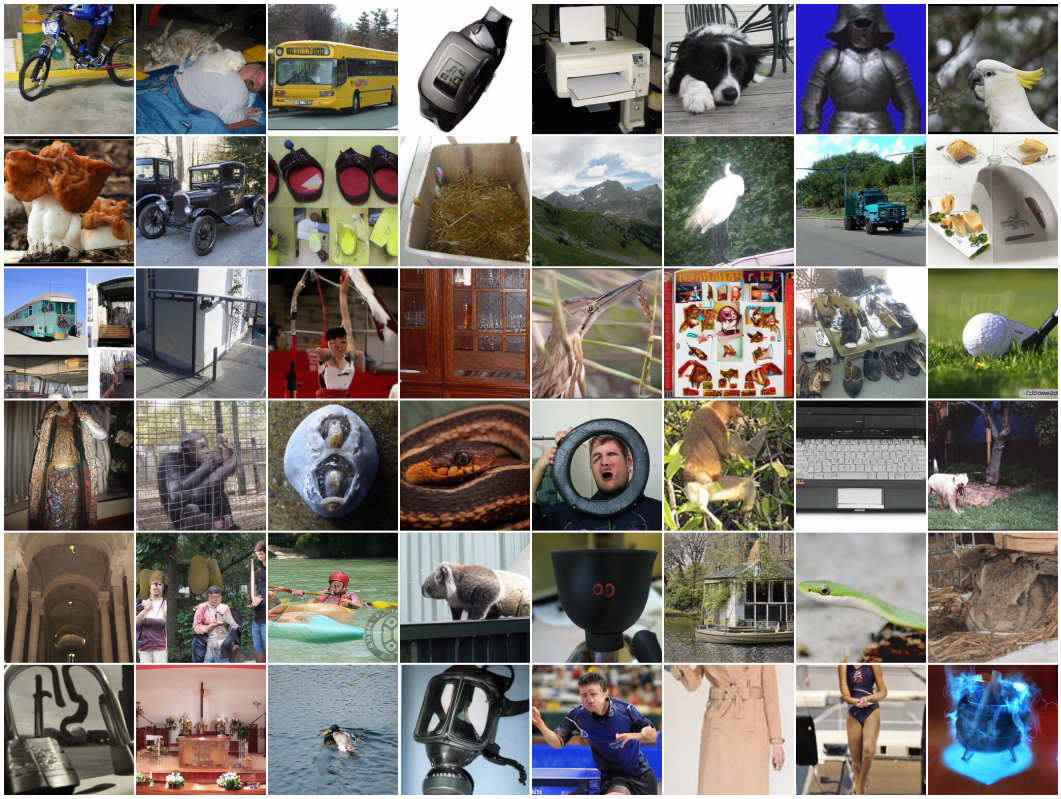

Figure 18: PoolDINO with 2×2 learned pooling ($4\times$ token compression), trained for 80 epochs, with $s_{\mathrm{IG}} = 1.75$ and no CFG. Samples are randomly selected without quality filtering.

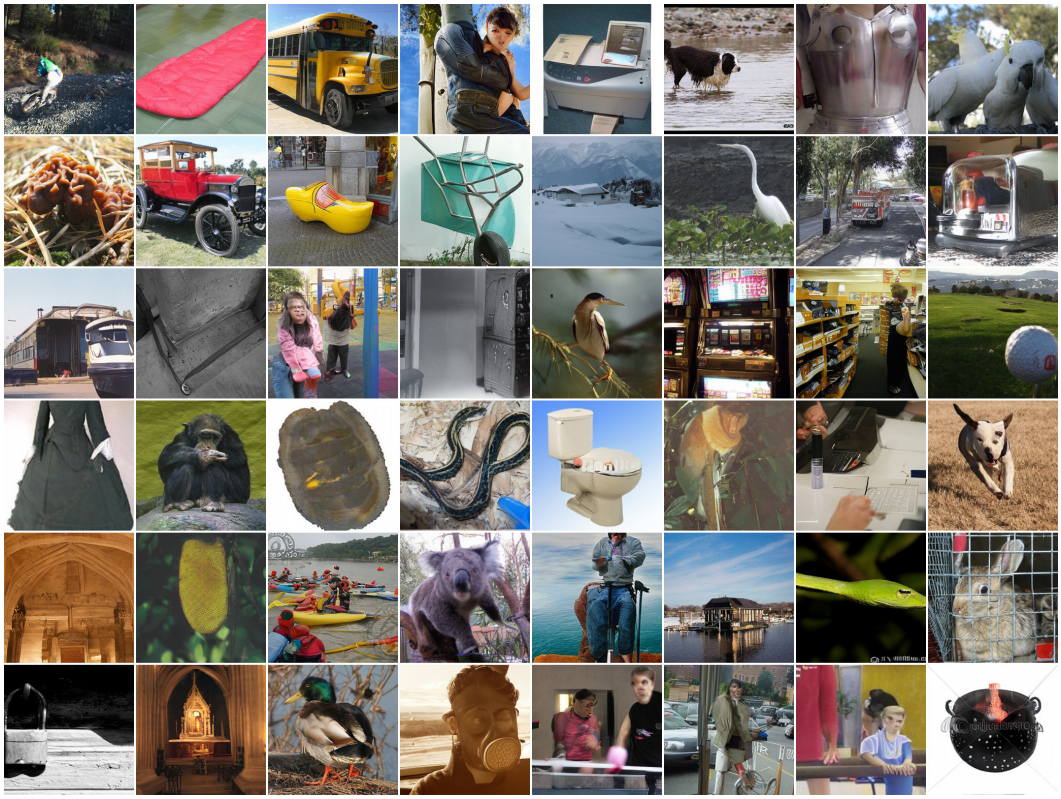

Figure 19: PoolDINO with 2×4 learned pooling ($8\times$ token compression), trained for 80 epochs, with $s_{\mathrm{IG}} = 2.00$ and no CFG. Samples are randomly selected without quality filtering.

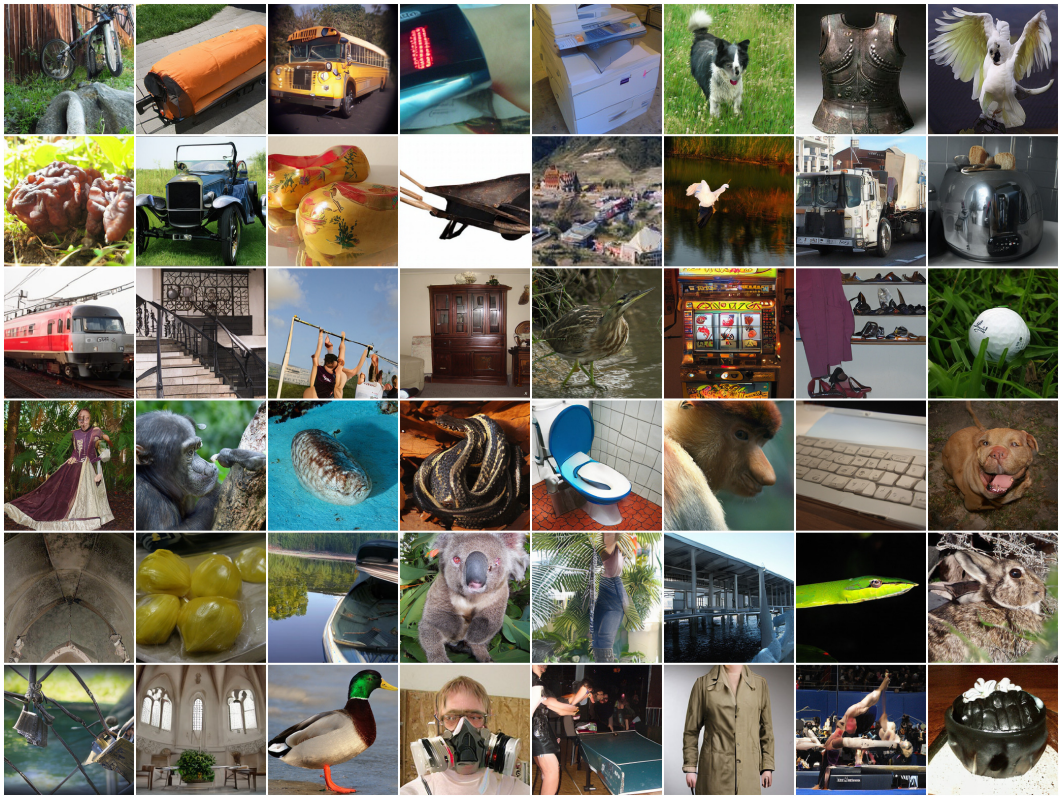

Figure 20: PoolDINO with 4×4 learned pooling ($16\times$ token compression), trained for 80 epochs, with $s_{\mathrm{IG}} = 2.75$ and no CFG. Samples are randomly selected without quality filtering.

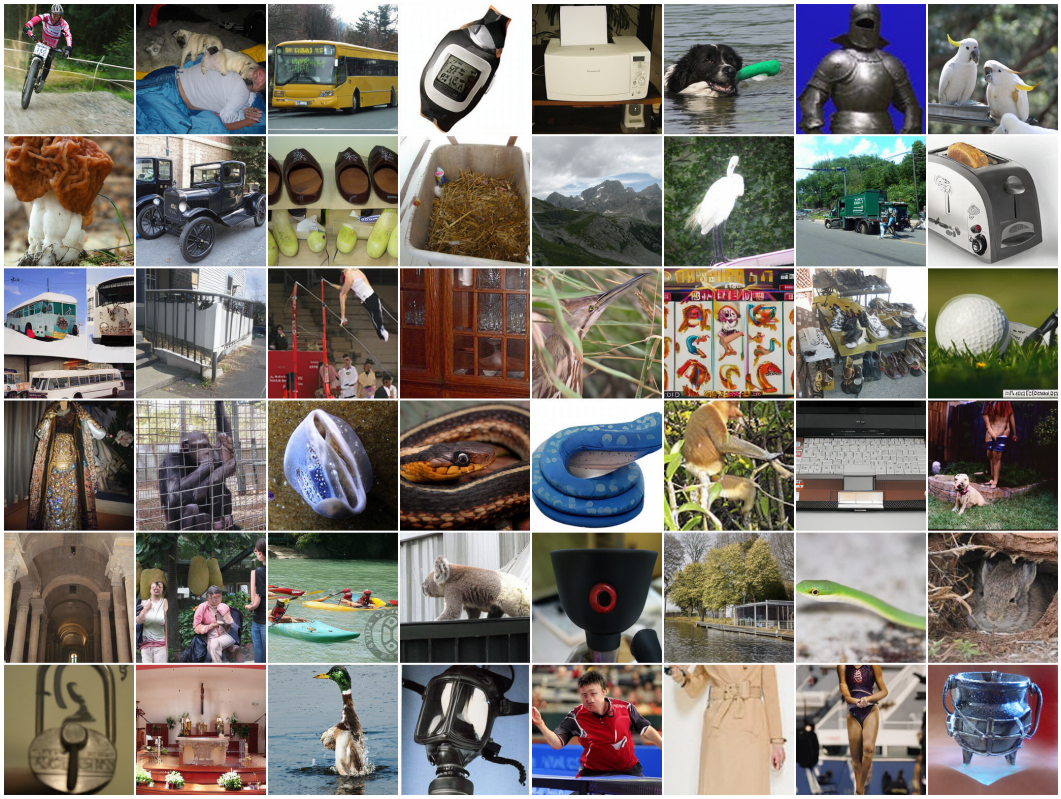

Figure 21: PoolDINO with 2×2 learned pooling and extended training ($4\times$ token compression), trained for 180 epochs, with $s_{\text{IG}} = 2.00$ and no CFG. Samples are randomly selected without quality filtering.

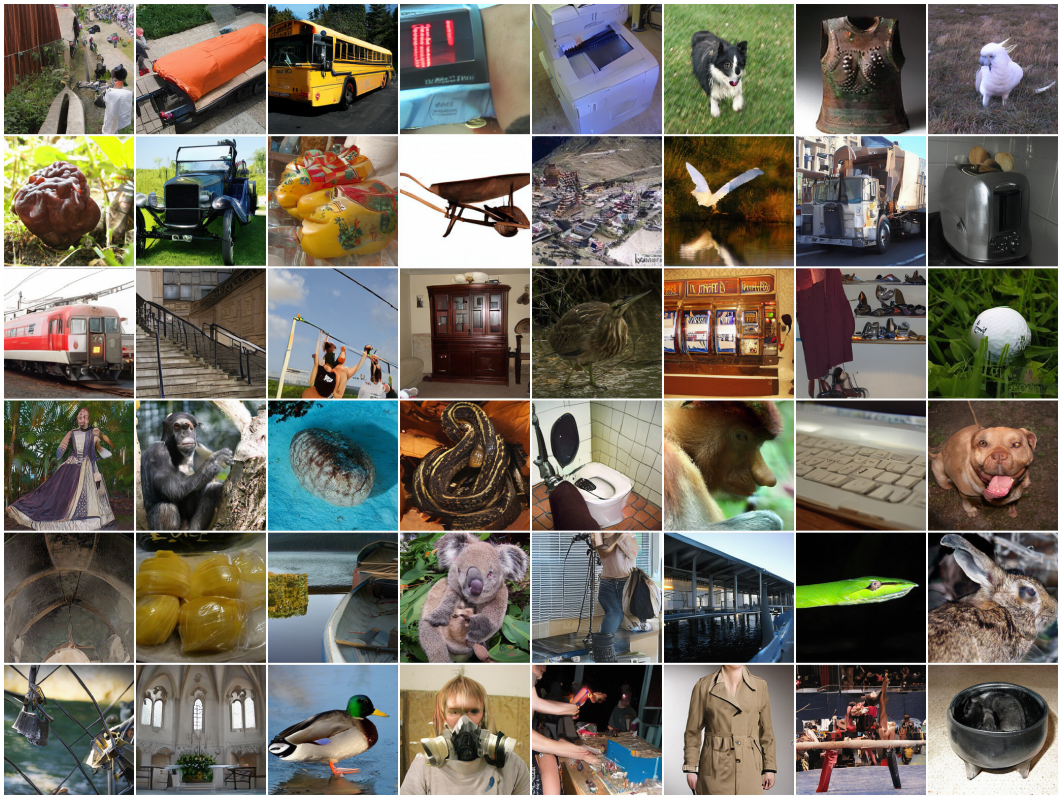

Figure 22: PoolDINO with 4×4 learned pooling and extended training ($16\times$ token compression), trained for 300 epochs, with $s_{\mathrm{IG}} = 2.75$ and no CFG. Samples are randomly selected without quality filtering.