RefineTok: Scale-Wise Tokenization with Progressive Visual Refinement

Yitian Zhang[1,2*] **Long Mai**[1] **Yizhou Wang**[2] **Yun Fu**[2]

[1]Adobe Research [2]Northeastern University

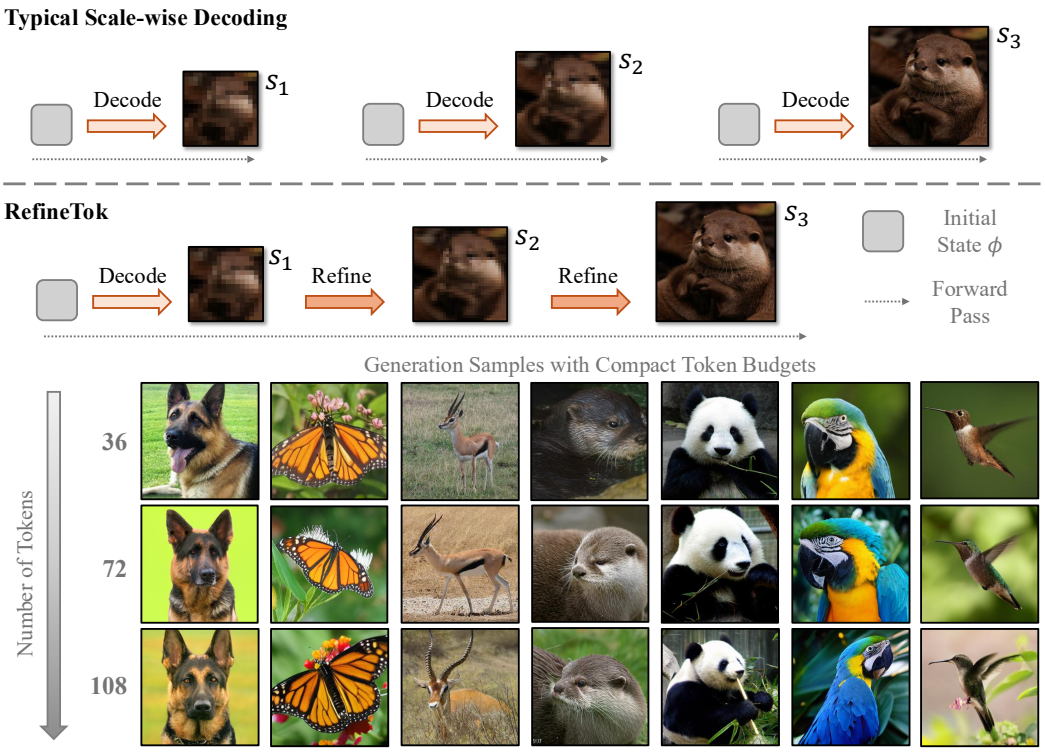


Figure 1: Comparison between a typical scale-wise decoding pattern and RefineTok. Conventional scale-wise tokenizers typically decode each target scale from an initial state, effectively modeling mappings from ϕ to s_i independently. In contrast, RefineTok models the transition between adjacent scales as progressive refinement, where each scale is obtained by refining the previous visual state. This design yields a structured coarse-to-fine latent interface and supports strong generation under compact token budgets, as illustrated by RefineTok with 36, 72, and 108 latent tokens.

Abstract

Image tokenization aims to compress visual signals into latent codes that remain effective for reconstruction and generation. Driven by the multi-scale nature of images, organizing latent along a coarse-to-fine refinement process emerges as a promising direction for visual tokenization. Existing scale-wise tokenizers often introduce this structure through token ordering or latent prefixes, but each target scale is typically decoded directly from its prefix, making different prefixes behave like separate reconstruction codes. This can lead to redundant encoding of global

*Work done when Yitian was an intern at Adobe Research.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

structure across scales and limits the effectiveness of compact token budgets. We propose **RefineTok**, a scale-wise continuous tokenizer that makes decoding itself progressive. RefineTok carries a visual state across scales and uses each latent prefix to refine the previous state into the next scale, encouraging scale-wise latents to act as incremental refinement signals rather than independent reconstruction codes. This design yields a compact and structured latent interface for both reconstruction and downstream generation. Together with prefix-level training, a coarse-centric scale trajectory, and a semantic anchor, RefineTok achieves strong reconstruction fidelity among scale-wise tokenizers and competitive class-conditional generation on ImageNet-1K under compact latent budgets. Meanwhile, it preserves a progressive scale-wise interface that single-scale tokenizers do not provide.

1 Introduction

Image tokenizers serve as the latent interface for representing visual signals in modern generative systems [14, 25, 31]. Early tokenizers typically preserve the spatial layout of images through 2D latent grids, including discrete tokenizers such as VQ-VAE and VQGAN [33, 27, 11], as well as continuous autoencoders used in latent diffusion models [18, 29]. While 2D grids provide a natural spatial representation, they often contain substantial redundancy for downstream sequence modeling. To obtain more compact latent interfaces, recent works [38, 7, 6, 39] have explored flat 1D tokenizers, where an image is compressed into a short sequence of visual tokens, and demonstrated that compact 1D latents can support strong reconstruction and downstream generation.

Beyond grid-based and flat latent representations, another promising direction is to organize visual tokens according to scale. Such a design matches the hierarchical structure of images, where global layout and object shape are captured at coarse resolutions, while local details and textures are refined at finer resolutions. This motivates scale-wise tokenization and generation, which model visual content along a coarse-to-fine trajectory. Recent works exploit this structure through next-scale prediction, ordered token prefixes, hierarchical residual decomposition, or flexible token schedules [32, 22, 40, 12, 1]. These methods show that scale-aware organization is an important design axis for visual tokenization and generation.

However, existing scale-wise tokenizers often exploit scale mainly through token ordering, grouping, or prediction order, while the decoding process itself remains less aligned with refinement. A common decoding strategy is to map each latent prefix directly to its target scale, so different prefixes behave like scale-specific reconstruction codes. This can force multiple prefixes to repeatedly encode global structure, layout, and object shape, rather than letting later groups focus on the missing information needed to refine a previous visual state. Consequently, the relation between adjacent scales is left mostly implicit, leading to redundant cross-scale representations and a less structured latent interface for downstream generation. In addition, most existing scale-wise tokenizers adopt discrete latent codes for autoregressive or next-scale prediction, which can limit reconstruction fidelity.

This limitation points to the need for explicitly modeling cross-scale transitions in the decoder. By carrying a visual state from one scale to the next, the decoder can refine what is missing at the current scale instead of repeatedly reconstructing global structure. As illustrated in Figure 1, this leads to a different decoding paradigm: the previous visual state is transformed into the next one through adjacent-scale refinement, rather than mapping each target scale separately from an initial state. Under this formulation, scale-wise latents provide refinement information for updating the carried visual state, resulting in less redundant cross-scale representation and a compact, structured interface for downstream generation.

Building on this insight, we present **RefineTok**, a scale-wise continuous tokenizer for progressive visual refinement. Specifically, we design a causal ViT encoder, which maps an image into coarse-to-fine latent groups, and introduce a progressive refinement decoder, which carries the visual state across scales and refines it with the corresponding latent prefix at each stage. To better match progressive refinement, RefineTok uses a coarse-centric trajectory that places more refinement stages at lower resolutions, where the global structure is first established. We further train the tokenizer with stochastic prefix truncation, so that sampled latent prefixes are optimized as valid intermediate representations rather than relying on future finer-scale latents. In addition, we introduce a DINOv2 [24]-aligned semantic anchor: a single training-time token that provides global semantic

guidance for organizing the scale-wise latent sequence, which is beneficial for both reconstruction and generation. Although RefineTok produces causal 1D latent groups, we show that a standard DiT-style [25] diffusion generator can effectively model this structured continuous latent space, achieving competitive class-conditional generation under compact token budgets. Besides, this progressive interface naturally exposes coherent intermediate states across scales, which single-scale tokenizers do not provide. Our contributions are summarized as follows:

- We propose RefineTok, a scale-wise continuous tokenizer that models images through progressive visual refinement, encouraging scale-wise latents to act as incremental refinement signals rather than independent reconstruction codes.
- We introduce a set of designs to support progressive refinement, including a progressive refinement decoder for adjacent-scale state transitions, a coarse-centric trajectory for stronger coarse visual states, stochastic prefix truncation for valid intermediate prefixes, and a DINOv2-aligned semantic anchor for global semantic guidance.
- We show that RefineTok achieves strong reconstruction fidelity among scale-wise tokenizers and competitive class-conditional generation under compact token budgets.

2 Related Work

2.1 Image Tokenizer

Single-scale tokenizers. Image tokenizers provide the latent interface for visual generation by compressing images into discrete or continuous representations. Early works largely preserve spatial structure with 2D latent grids, including discrete tokenizers such as VQ-VAE [33, 27], VQGAN [11], and improved variants [2, 19], as well as continuous autoencoders such as VAEs and KL-regularized autoencoders [18, 29]. These continuous 2D latents have become a standard interface for latent diffusion models, and recent autoencoding designs further improve reconstruction quality and latent efficiency [26, 8, 41]. Beyond 2D grids, flat 1D tokenizers compress images into short latent sequences to reduce spatial redundancy. Representative examples include TiTok [38], as well as recent continuous 1D tokenizers such as SoftVQ-VAE [7], MAETok [6], and TexTok [39], which demonstrate strong reconstruction and generation performance with diffusion-based generators.

Scale-wise tokenizers. Another line of work introduces coarse-to-fine, hierarchical, or scale-wise structure into visual tokenization and generation. A representative direction is next-scale prediction [32], where images are generated from coarse to fine. Follow-up works improve it through more flexible token schedules, stronger scale-wise tokenizers, or enhanced next-scale generators, including FlexVAR [17], ImageFolder [21], NFIG [16], and NVG [35]. In parallel, 1D scale-wise or ordered tokenizers have been explored from different perspectives, including variable-length tokenization [1], next-detail prediction [22], frequency-ordered generation [15], hierarchical residual tokenization [40], and balanced ordered token usage [12]. Recent works further combine scale-wise or structured latent representations with flow- or diffusion-based generation, such as FlowAR [28], FAR [36], and HieraTok [5]. These methods show that scale-aware organization is another important design axis.

2.2 Visual Generation

Autoregressive models. Autoregressive image generation critically depends on how visual tokens are organized into a prediction sequence. Early token-based approaches typically linearize 2D token grids using raster scan order, enabling next-token prediction with decoder-only transformers [37, 31, 34]. VAR [32] replaces rasterized prediction with next-scale prediction, and subsequent methods improve scale-wise generation through flexible scale schedules, stronger tokenizers, or enhanced generators [17, 21, 16, 35]. For 1D token sequences, FlexTok [1], DetailFlow [22], SpectralAR [15], ResTok [40], and ReTok [12] further show that ordered or scale-aware token organization can benefit autoregressive generation.

Diffusion models. Diffusion models [14] synthesize data by iteratively denoising samples from a noise distribution, and are now widely adopted for high-quality image generation. Latent diffusion models [29] show that diffusion can be performed efficiently in learned latent spaces, and DiT [25] further demonstrates the scalability of transformer-based diffusion generators. Recent continuous tokenizers, including SoftVQ-VAE [7], MAETok [6], and TexTok [39], are often paired with DiT-style generators, obtaining competitive generation performance. Hybrid autoregressive-diffusion

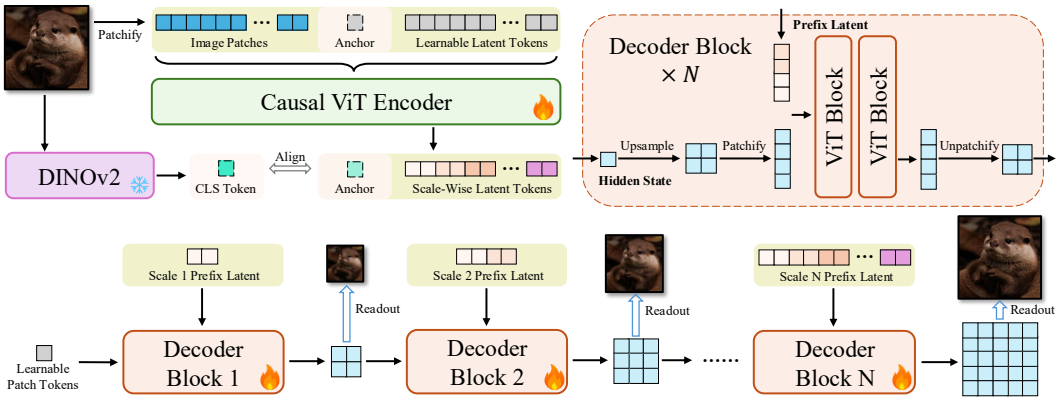


Figure 2: Architecture of RefineTok. An input image is patchified and processed by a causal ViT encoder along with a training-time semantic anchor token and learnable latent tokens. The semantic anchor is aligned with a pretrained DINOv2 feature, while the encoder outputs scale-wise latent tokens arranged in a coarse-to-fine order. For decoding, RefineTok adopts a progressive refinement decoder: at each scale, the corresponding latent prefix conditions a decoder block that refines an intermediate hidden state propagated from the previous stage. The hidden state is progressively upsampled across stages, enabling coarse-to-fine reconstruction with intermediate readout at every scale.

models such as MAR [20], FlowAR [28], and FAR [36] also combine structured token prediction with diffusion or flow-based refinement. Our generator follows a standard DiT-style diffusion setup.

3 Methodology

Figure 2 provides an overview of RefineTok. We first describe the causal encoder that produces scale-wise continuous latents, then introduce the progressive refinement decoder that propagates visual states across scales. We further present the coarse-centric trajectory, stochastic prefix truncation, and semantic anchor used to support stable scale-wise refinement, followed by the training objective.

3.1 Causal Encoder

RefineTok uses a ViT encoder to produce scale-wise continuous latent tokens. Given an input image x, we first patchify it into image tokens and append a set of learnable latent tokens. The encoder is a ViT operating on the concatenated sequence. To inject spatial structure, we apply 2D rotary positional embeddings (RoPE) [30] to both patch tokens and latent tokens. Since RefineTok is scale-wise, latent groups are arranged in a coarse-to-fine order and processed with a scale-aware causal attention mask. Specifically, patch tokens attend fully to each other, while latent tokens at scale s can only attend to patch tokens and latent groups from scales $\leq s$. This prevents earlier latent groups from depending on future finer-scale latents, encouraging them to remain coarse. After passing the sequence through the encoder blocks, we read out the latent portion and project it to the latent space:

$$\{z_1, z_2, \ldots, z_S\} = \mathrm{Enc}(x), \tag{1}$$

where each $z_s \in \mathbb{R}^{N_s \times C}$ contains N_s latent tokens of dimension C for scale s.

3.2 Progressive Refinement Decoder

Prior scale-wise tokenizers organize latents in a coarse-to-fine order, but often decode each target scale directly from an initial visual state. Given a latent prefix $\mathbf{z}_{\leq s}$, this can be written as

$$\hat{x}_s = D_s(\phi, \mathbf{z}_{\leq s}), \tag{2}$$

where ϕ denotes the initial state (learnable patch tokens). This treats different scales as separate reconstruction targets, so each prefix may need to recover global structure and layout repeatedly.

RefineTok instead decodes through progressive refinement. Starting from $h_0 = \phi$, each stage carries over the previous visual state and refines it with the current latent prefix:

$$h_s = D_s(h_{s-1}^{\uparrow}, \mathbf{z}_{\leq s}), \qquad \hat{x}_s = R_s(h_s), \tag{3}$$

where $h_{s-1}^{\uparrow}$ is the upsampled previous state, D_s is the refinement block at scale s, and R_s maps the updated state to the reconstruction at scale s. Thus, RefineTok models the transition $s_{i-1} \to s_i$ rather than repeatedly mapping from ϕ to each target scale. This encourages scale-wise latents to act as incremental refinement signals rather than independent reconstruction codes, yielding a more structured generator-facing latent sequence.

In practice, each refinement stage is implemented as a small stack of ViT blocks. The carried-over visual state is maintained as spatial feature maps and is first upsampled to the next resolution using interpolation followed by a convolutional projection. The upsampled state is then patchified into visual tokens, concatenated with the projected latent prefix as conditioning tokens, and processed by the ViT blocks through joint self-attention. After each stage, the updated visual tokens are unpatchified back to a spatial state and decoded by a lightweight readout head into the current-scale reconstruction. The updated state is then propagated to the next stage. Since the resolution increases progressively, RefineTok produces valid intermediate reconstructions at all scales in a single forward pass without running a full-resolution decoder independently for each target scale.

3.3 Coarse-Centric Trajectory

Since the decoder is progressive and each stage refines the visual state propagated from earlier ones, the quality of later stages strongly depends on the quality of earlier coarse stages. If the coarse reconstruction is weak, later stages must spend substantial capacity correcting global structure before adding finer details. Motivated by this observation, we allocate more refinement capacity to smaller scales by using a denser trajectory in the coarse regime. This design better matches the progressive refinement structure of the decoder: early stages focus on building a reliable coarse representation, and later stages can then refine it with relatively few additional latents.

3.4 Semantic Anchor

We introduce a semantic anchor during tokenizer training. Concretely, we prepend a single extra token before all latent tokens, so that it appears at the beginning of the latent sequence and acts as the coarsest semantic token. This token is trained to align with a pretrained DINOv2 [24] global feature of the input image, and is processed together with the other latent tokens by the encoder. Note that only this single token is aligned with DINOv2; the scale-wise latent tokens are never directly aligned with pretrained features, and the anchor is not used by the downstream generator. Since RefineTok organizes latent tokens in a coarse-to-fine order, placing the extra token at the front of the latent sequence encourages it to capture global semantic information before finer-scale latent tokens describe progressively more detailed refinements. In this way, the latent sequence is anchored by a stable coarse semantic representation, which improves the overall quality of the learned latent space.

3.5 Stochastic Prefix Truncation

Since the decoder is progressive, a single forward pass can naturally produce intermediate reconstructions at multiple scales. A straightforward choice would therefore be to supervise all scales simultaneously. However, this introduces competing objectives across scales and tends to make the latent space heterogeneous, with different latent groups specializing to different scale-specific targets.

To avoid this, we adopt stochastic prefix truncation. For each training example, we draw a random target scale s, keep only the latent prefix $\mathbf{z}_{\leq s}$, and compute the reconstruction loss solely at that scale. Let x_s denote the ground-truth image resized to scale s. We optimize

$$\hat{x}_s = \mathrm{Dec}(\mathbf{z}_{\leq s}), \qquad \mathcal{L}_{\mathrm{rec}} = \ell_{\mathrm{rec}}(\hat{x}_s, x_s), \tag{4}$$

where $s \sim p(s)$ is sampled during training. In practice, we assign 80% probability to the largest scale and distribute the remaining 20% uniformly over the other scales. Since decoding $\mathbf{z}_{\leq s}$ passes through all stages up to s, every latent group and refinement stage still receives gradients in at least 80% of the training steps. This training strategy makes each prefix $\mathbf{z}_{\leq s}$ responsible for its own scale, without relying on future finer-scale latents. As a result, the latent space becomes more homogeneous across scales, and each prefix is encouraged to represent a valid intermediate refinement state.

Table 1: System-level comparison on ImageNet-1K 256×256 in terms of reconstruction and class-conditional generation.

Reconstruction					Generation						
Tokenizer	Type	# tokens	PSNR↑	rFID↓	Generator	Type	Param.	gFID↓	IS↑	Precision↑	Recall↑
Discrete modeling											
Single-scale											
VQGAN [4]	2D	256	18.15	2.28	MaskGIT [4]	Mask.	227M	6.18	182.1	0.80	0.51
LlamaGen [31]	2D	256	20.79	2.19	GPT-XL [31]	AR	775M	2.62	244.1	0.80	0.57
TiTok-S [38]	1D	128	17.72	1.71	MaskGIT [4]	Mask.	177M	1.97	281.8	–	–
Scale-wise											
VAR [32]	2D	680	21.30	0.90	VAR-d20 [32]	VAR	600M	2.95	302.6	0.83	0.56
FlexVAR [17]	2D	680	–	–	VAR-d20 [17]	VAR	600M	2.95	302.6	0.83	0.56
ImageFolder [21]	2D	572	–	0.80	VAR-GPT [32]	VAR	362M	2.60	332.4	0.77	0.59
FR-VAE [16]	2D	680	–	0.85	NFIG [16]	VAR	310M	2.81	295.0	0.75	0.63
NVG-VAE [35]	2D	511	–	0.74	NVG-d20 [35]	VAR	622M	2.44	310.4	0.80	0.60
FlexTok [1]	1D	128	15.90	1.20	LlamaGen [31]	AR+Diff.	1.33B	1.89	275.6	0.82	0.61
DetailFlow [22]	1D	128	19.46	1.22	GPT-L [31]	AR	326M	2.96	221.4	0.82	0.57
SpectralAR [15]	1D	64	–	4.03	SpectralAR-d20 [15]	AR	600M	2.49	305.4	–	–
ResTok [40]	1D	128	–	1.28	HAR [40]	AR	326M	2.34	257.8	0.79	0.60
ReTok [12]	1D	256	20.57	1.01	LlamaGen-XL [31]	AR	775M	2.27	245.9	0.82	0.60
Continuous modeling											
Single-scale											
LDM-VAE [29]	2D	4096	27.53	0.27	LDM-4 [29]	Diff.	400M	3.60	247.7	0.87	0.48
SDXL-VAE [29]	2D	1024	26.04	0.62	DiT-XL/2 [25]	Diff.	675M	2.27	278.2	0.83	0.57
SDXL-VAE [29]	2D	1024	26.04	0.62	SiT-XL/2 [23]	Diff.	675M	2.06	270.3	0.82	0.59
MAR-VAE [20]	2D	256	26.18	0.53	MAR-L [20]	AR+Diff.	479M	1.78	296.0	0.81	0.60
SoftVQ-B [7]	1D	64	–	0.88	DiT-XL [25]	Diff.	675M	3.29	262.5	0.84	0.54
TexTok [39]	1D	128	22.05	1.04	DiT-XL [25]	Diff.	675M	1.66	294.4	0.80	0.61
MAETok [6]	1D	128	23.61	0.48	DiT-XL [25]	Diff.	675M	1.73	308.4	0.80	0.63
Scale-wise											
MAR-VAE [20]	2D	341	26.18	0.53	FlowAR-L [28]	AR+Diff.	589M	1.90	281.4	0.83	0.57
MAR-VAE [20]	2D	256	26.18	0.53	FAR-H [36]	AR+Diff.	812M	3.21	300.6	0.81	0.55
HieraTok [5]	2D	341	–	0.45	DiT [25]	Diff.	–	1.82	–	–	–
RefineTok	1D	36	20.87	1.60	DiT-XL	Diff.	675M	2.68	245.2	0.84	0.55
RefineTok	1D	72	21.65	1.04	DiT-XL	Diff.	675M	2.03	253.1	0.83	0.59
RefineTok	1D	108	22.85	0.76	DiT-XL	Diff.	675M	1.74	267.2	0.82	0.60

3.6 Training Objective

We train RefineTok with the standard tokenizer losses commonly used in prior works, namely pixel reconstruction, perceptual, and adversarial losses. Besides, we apply the semantic alignment loss on the semantic anchor described above. The overall objective is written as:

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + \lambda_1 \mathcal{L}_{\text{percep}} + \lambda_2 \mathcal{L}_{\text{adv}} + \lambda_3 \mathcal{L}_{\text{sem}}, \tag{5}$$

where $\mathcal{L}_{\text{rec}}$, $\mathcal{L}_{\text{percep}}$, $\mathcal{L}_{\text{adv}}$, and $\mathcal{L}_{\text{sem}}$ denote the reconstruction, perceptual, adversarial, and semantic alignment losses, respectively, and $\lambda_1, \lambda_2, \lambda_3$ are balancing weights.

4 Experiments

4.1 Experimental Setup

Implementation Details. RefineTok uses a scale-wise latent trajectory with 9 scales by default, with resolutions $[16, 24, 32, 48, 64, 96, 128, 192, 256]$, and the latent tokens are evenly divided across scales, *i.e.*, 4, 8, and 12 tokens per scale for the 36-, 72-, and 108-token budgets. Both the encoder and decoder are built from ViT blocks (width 768, 12 heads). The encoder has 16 layers, each decoder block has 2 layers, and each latent token has 16 channels. For semantic anchoring, we align the training-time anchor token with the global feature of a pretrained DINOv2-Base model [24]. The adversarial loss is implemented with a DINO-based discriminator [3] and the loss weights are set to $\lambda_1 = 1.0$, $\lambda_2 = 0.75$, and $\lambda_3 = 0.02$. For generation experiments, we adopt a DiT-style [25] latent diffusion generator operating directly on the RefineTok latent sequence. Please see Appendix for more details.

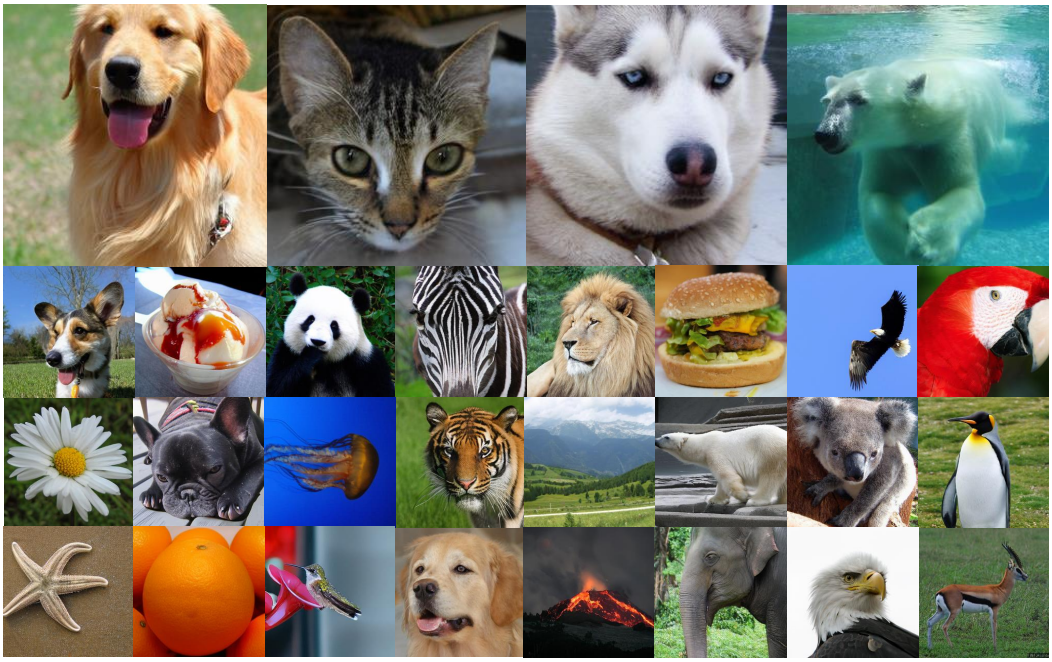


Figure 3: Class-conditional samples on ImageNet 256×256 from RefineTok-108 with DiT-XL. The samples are semantically meaningful and exhibit delicate fine-grained details.

Evaluation Protocol. We assess reconstruction and downstream generation on ImageNet-1K [9], following prior methods [22, 39]. For reconstruction, we measure PSNR and reconstruction FID (rFID) [13] on the ImageNet validation split. For class-conditional generation, we sample 50K images and compute generation FID (gFID), Inception Score (IS), precision, and recall. Unless otherwise specified, gFID, IS, precision, and recall are computed using the standard ImageNet statistics and evaluation pipeline used in ADM [10]. For intermediate scales, each reconstruction is compared with the ground truth resized to the corresponding resolution, or bicubically upsampled and compared with the original 256×256 image.

4.2 Main Results

Table 1 compares RefineTok with representative discrete and continuous tokenizers on ImageNet-1K 256×256. For reconstruction, RefineTok achieves strong fidelity among scale-wise tokenizers while using a compact 1D latent sequence. Most prior scale-wise tokenizers are based on discrete latents, whose reconstruction fidelity is relatively limited. In contrast, RefineTok substantially improves PSNR over these scale-wise baselines. Figure 4 compares intermediate reconstruction trajectories of representative 1D scale-wise tokenizers. RefineTok produces a smoother and more coherent coarse-to-fine trajectory in a single decoder forward pass, while FlexTok [1] and DetailFlow [22] require separate forward passes to obtain different-scale outputs and generate less consistent trajectory. Although some single-scale continuous tokenizers achieve higher reconstruction scores with larger latent capacity (*e.g.*, MAETok uses 128 tokens of dimension 32, while RefineTok uses 108 tokens of dimension 16), RefineTok provides a favorable trade-off between compactness and reconstruction fidelity in the scale-wise setting.

On downstream generation, RefineTok remains competitive under the DiT-XL backbone. In particular, RefineTok-72 already surpasses the DiT-XL/2 baseline using SDXL-VAE, although the latter operates on 256 tokens after DiT patchification. RefineTok-108 further achieves a gFID of 1.74 with only 108 latent tokens, comparing favorably with prior strong single-scale tokenizers [6, 39]. Figure 3 shows that RefineTok-108 produces diverse and semantically meaningful samples with clear object structure and fine-grained visual details. Moreover, gFID improves consistently from 2.68 to 2.03 and 1.74 as the token budget increases from 36 to 72 and 108, showing that RefineTok scales well with the latent budget. Besides, RefineTok provides a structured coarse-to-fine interface with intermediate states and enables progressive reconstruction and generation trajectories from a single decoder forward pass, which is not available in single-scale tokenizers.

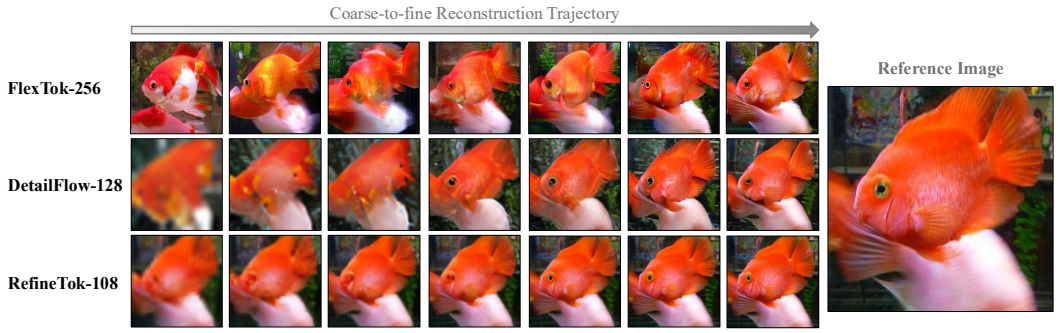


Figure 4: Comparison of reconstruction trajectories for representative 1D scale-wise tokenizers. FlexTok and DetailFlow require separate decoder forward passes to obtain different-scale outputs, whereas RefineTok produces the full trajectory in a single forward pass. RefineTok also yields a smoother and more coherent trajectory, with more stable visual structure across intermediate scales.

Table 2: Ablation study of RefineTok. The first row is a flat tokenizer baseline without scale-wise modeling, PRD and CCT denote Progressive Refinement Decoder and Coarse-Centric Trajectory, respectively. GFLOPs reports the decoder cost of one forward pass at the largest scale, while GFLOPs (all) reports the decoder cost of obtaining the full trajectory. Without PRD, each prefix is decoded independently, requiring 9 forward passes for the full trajectory; with PRD, all scales are obtained in a single forward pass.

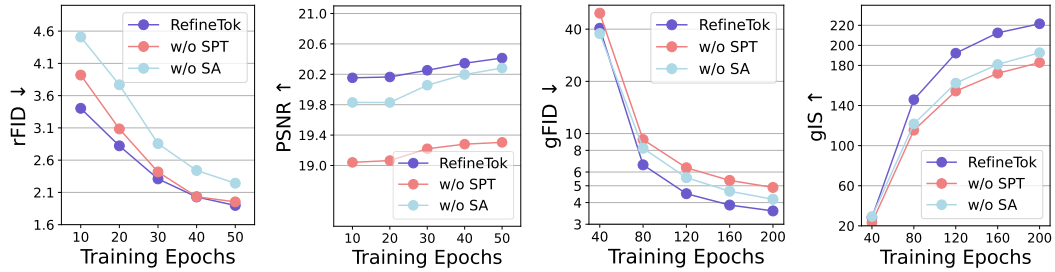
Scale-wise	PRD	CCT	GFLOPs	GFLOPs (all)	PSNR↑	rFID↓	gFID↓	gIS↑
✗	✗	✗	202.7	–	20.72	1.81	7.96	119.3
✓	✗	✓	202.7	441.6	20.24	1.77	7.26	134.6
✓	✓	✗	75.7	75.7	19.30	1.94	7.71	129.4
✓	✓	✓	50.6	50.6	20.41	1.89	6.59	145.9

4.3 Design Choices of RefineTok

Scale-wise Prior. We first compare RefineTok with a flat tokenizer baseline that removes the scale-wise structure. In this variant, all latent tokens are treated as a single monolithic sequence, and the decoder reconstructs the image directly at the final resolution without any scale-wise prior. As shown in Table 2, this baseline achieves slightly better reconstruction quality than the full model. However, its downstream generation performance is substantially worse, with gFID increasing from 6.59 to 7.96. This indicates that scale-wise organization provides a more structured latent interface for the generator, which remains useful even when the downstream generator is diffusion-based rather than autoregressive. In addition, since the monolithic decoder operates at the largest resolution throughout the entire decoding process, its decoder cost is also much higher than our progressive decoder under the same model capacity.

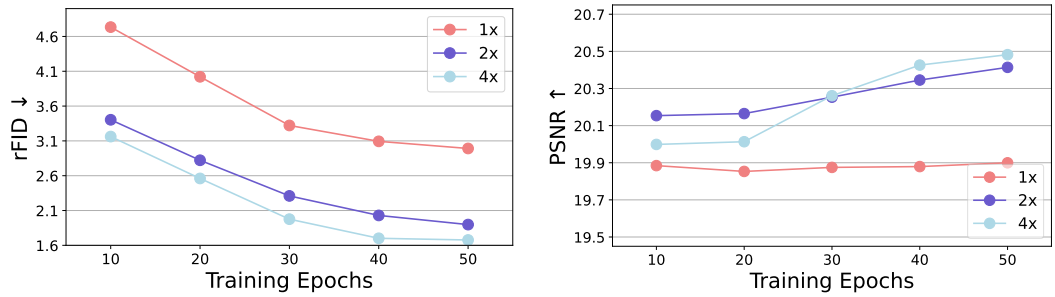
Progressive Refinement Decoder. We next ablate the Progressive Refinement Decoder by replacing it with a shared full decoder similar to DetailFlow [22]. In this variant, different latent prefixes are still used for different target scales, but each prefix is decoded by passing through the full decoder from scratch, rather than refining a carried-over state from the previous scale. This design has a similar single-scale decoding cost to the monolithic decoder, and obtaining the full trajectory requires running the decoder multiple times, leading to a much larger GFLOPs (all). Although its reconstruction quality is comparable to RefineTok, its generation performance is clearly worse. This suggests that progressive state carry-over is not only a computationally efficient decoding strategy, but also helps align the tokenizer with the generator-facing scale-wise latent structure.

Coarse-Centric Trajectory. We further study the effect of the scale trajectory. Our default design uses a coarse-centric trajectory $[16, 24, 32, 48, 64, 96, 128, 192, 256]$, which allocates more refinement stages to smaller scales. We compare it with a large-scale-biased trajectory $[16, 32, 64, 96, 128, 160, 192, 224, 256]$, which places more stages near higher resolutions. Removing the coarse-centric design leads to a clear drop in reconstruction quality. This is consistent with the progressive nature of RefineTok: early coarse stages establish the global visual state, and later stages can only refine what has already been built. If the low-resolution states are underdeveloped, later



(a) rFID over tokenizer training epochs. (b) PSNR over tokenizer training epochs. (c) gFID over generator training epochs. (d) gIS over generator training epochs.

Figure 5: Ablation on Stochastic Prefix Truncation (SPT) and Semantic Anchor (SA). Panels (a) and (b) show reconstruction performance over tokenizer training epochs, while panels (c) and (d) report generation performance over generator training epochs.



(a) rFID over tokenizer training epochs. (b) PSNR over tokenizer training epochs.

Figure 6: Ablation on the number of ViT blocks per refinement stage. Panels (a) and (b) report rFID and PSNR over tokenizer training epochs, respectively.

high-resolution stages must spend capacity correcting global structure, making the overall refinement process less effective. This variant also has a higher decoder cost because more refinement computation is performed at larger resolutions. The generation drop further suggests that the trajectory design affects not only reconstruction, but also the generator-facing latent distribution: a weak coarse stage makes later latents compensate for missing global structure, resulting in a less stable and harder-to-model scale-wise sequence.

Stochastic Prefix Truncation. We ablate Stochastic Prefix Truncation (SPT) by decoding each scale only from its own scale-specific latent group and supervising all preceding sampled-scale outputs during tokenizer training. As shown in Figure 5, this variant obtains slightly worse rFID than RefineTok, but its PSNR is substantially lower, indicating weaker reconstruction fidelity. This suggests that jointly imposing multiple scale-specific objectives can introduce conflicting supervision and make the latent groups specialize to their own scales rather than forming a coherent prefix representation. The effect is more pronounced in downstream generation: the generator trained on this latent space yields the worst gFID and gIS among the compared variants. This supports the importance of SPT for learning a more consistent generator-facing latent interface.

Semantic Anchor. Shown in Figure 5, removing the DINOv2-aligned Semantic Anchor (SA) consistently hurts both reconstruction and generation. The anchor acts as a lightweight global semantic reference at the beginning of the coarse-to-fine latent sequence, encouraging the remaining scale-wise latents to organize around a stable coarse semantic representation. Although the anchor is only used during tokenizer training, it yields a better latent space and stronger downstream generation. SA is also not tied to DINOv2: aligning the anchor with CLIP or SigLIP2 likewise improves over the variant without SA (Appendix A.4).

Decoder depth. Figure 6 studies how many ViT blocks each refinement stage uses. Increasing the stage depth from 1x to 2x substantially improves reconstruction quality, showing that sufficient per-stage capacity is important for progressive refinement. Using 4x blocks further improves rFID and eventually surpasses 2x in PSNR as training proceeds, although the overall gain over 2x remains modest. Considering the significantly increased model size and computation of 4x, we use 2 ViT blocks per stage by default as a balanced choice between reconstruction fidelity and efficiency.

Table 3: Per-scale reconstruction PSNR of RefineTok-108. Each intermediate reconstruction is compared with the ground truth resized to its own resolution, or with the original 256×256 image after bicubic upsampling.

Resolution	16	24	32	48	64	96	128	192	256
PSNR↑ (resized GT)	24.83	24.63	24.66	24.34	24.76	23.85	24.18	23.68	22.85
PSNR↑ (256 GT)	17.75	18.60	19.23	19.98	20.79	21.14	21.83	22.41	22.85

Table 4: Linear probing on cumulative latent prefixes of frozen RefineTok-108.

# tokens	**Top-1↑**	**Top-5↑**
12	9.44	43.30
36	9.94	45.58
72	10.00	46.49
108	10.02	47.06

4.4 Analysis of Progressive Refinement

Per-scale reconstruction. Table 3 evaluates the reconstruction of RefineTok-108 at every scale. Compared with the correspondingly resized ground truth, all intermediate reconstructions maintain stable PSNR, indicating that every prefix forms a valid reconstruction state. Compared with the full-resolution image, PSNR increases monotonically from 17.75 to 22.85, showing that each latent group consistently refines the carried-over visual state.

Linear probing. With the tokenizer frozen, we fit a linear classifier on cumulative latent prefixes. As shown in Table 4, the first 12 tokens already capture most of the linearly accessible class information, and extending the prefix to 108 tokens brings only marginal gains. This indicates that global semantics are established early, while later latent groups mainly contribute finer refinements rather than re-encoding class-level information. Note that the absolute accuracy is limited by the 16-dimensional latents of a reconstruction tokenizer. Consistently, the reconstruction updates between adjacent scales shift from low- to high-frequency content along the trajectory (Appendix A.3).

Why traverse all stages? Since the full latent sequence contains information from all scales, it could in principle be decoded directly at the final resolution, as in the flat baseline and the shared full decoder in Table 2. However, such a decoder can jointly reinterpret all latent groups at the final scale, without requiring later groups to build on the coarse state established earlier. RefineTok instead constructs the final state progressively, so that each latent group refines what previous groups have built. As shown in Table 2, both alternatives achieve comparable reconstruction but worse generation, and they are more costly even for final-scale reconstruction alone (202.7 *vs.* 50.6 GFLOPs), since most refinement stages of RefineTok operate at coarse resolutions.

5 Discussion

Conclusion. We presented RefineTok, a scale-wise continuous tokenizer for progressive visual refinement. Instead of decoding each target scale directly from its latent prefix, RefineTok carries a visual state across scales and models adjacent-scale refinement, encouraging scale-wise latents to act as incremental refinement signals. This design reduces redundant cross-scale representation and provides a compact, structured latent interface for reconstruction and downstream generation. With prefix-level training, a coarse-centric scale trajectory, and a semantic anchor, RefineTok achieves strong reconstruction fidelity among scale-wise tokenizers and competitive class-conditional generation on ImageNet-1K under compact token budgets, while preserving a progressive scale-wise interface.

Limitation. Due to computational constraints, we have not fully explored the scalability of RefineTok across different model sizes. In addition, the design of the refinement trajectory is not exhaustively studied. For example, different choices of resolution schedules, number of scales, and number of tokens allocation per scale may further affect the performance. Moreover, RefineTok currently trains all stages jointly; a progressive freezing strategy that appends new stages while preserving earlier ones would require a more modular encoder, which we leave for future work. We also leave the discrete version of RefineTok unexplored. When paired with an autoregressive generator, the discrete RefineTok could enable preview-style generation by producing intermediate results along the progressive trajectory.

References

[1] R. Bachmann, J. Allardice, D. Mizrahi, E. Fini, O. F. Kar, E. Amirloo, A. El-Nouby, A. Zamir, and A. Dehghan. Flextok: Resampling images into 1d token sequences of flexible length. In *ICML*, 2025.

[2] S. Cao, Y. Yin, L. Huang, Y. Liu, X. Zhao, D. Zhao, and K. Huang. Efficient-vqgan: Towards high-resolution image generation with efficient vision transformers. In *ICCV*, 2023.

[3] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021.

[4] H. Chang, H. Zhang, L. Jiang, C. Liu, and W. T. Freeman. Maskgit: Masked generative image transformer. In *CVPR*, 2022.

[5] C. Chen, Z. Huang, C. Zou, M. Zhu, K. Ji, J. Liu, J. Chen, H. Chen, and C. Shen. Hieratok: Multi-scale visual tokenizer improves image reconstruction and generation. *arXiv preprint arXiv:2509.23736*, 2025.

[6] H. Chen, Y. Han, F. Chen, X. Li, Y. Wang, J. Wang, Z. Wang, Z. Liu, D. Zou, and B. Raj. Masked autoencoders are effective tokenizers for diffusion models. In *ICML*, 2025.

[7] H. Chen, Z. Wang, X. Li, X. Sun, F. Chen, J. Liu, J. Wang, B. Raj, Z. Liu, and E. Barsoum. Softvq-vae: Efficient 1-dimensional continuous tokenizer. In *CVPR*, 2025.

[8] J. Chen, H. Cai, J. Chen, E. Xie, S. Yang, H. Tang, M. Li, Y. Lu, and S. Han. Deep compression autoencoder for efficient high-resolution diffusion models. *arXiv preprint arXiv:2410.10733*, 2024.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[10] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *NeurIPS*, 2021.

[11] P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021.

[12] Z. Fu, L. Guo, C. Wang, B. Song, D. Liu, and B. Wen. Improving flexible image tokenizers for autoregressive image generation. *arXiv preprint arXiv:2601.01535*, 2026.

[13] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS*, 2017.

[14] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 2020.

[15] Y. Huang, W. Chen, W. Zheng, Y. Duan, J. Zhou, and J. Lu. Spectralar: Spectral autoregressive visual generation. In *ICCV*, 2025.

[16] Z. Huang, X. Qiu, Y. Ma, Y. Zhou, J. Chen, H. Zhang, C. Zhang, and X. Li. Nfig: multi-scale autoregressive image generation via frequency ordering. *arXiv preprint arXiv:2503.07076*, 2025.

[17] S. Jiao, G. Zhang, Y. Qian, J. Huang, Y. Zhao, H. Shi, L. Ma, Y. Wei, and Z. Jie. Flexvar: Flexible visual autoregressive modeling without residual prediction. *arXiv preprint arXiv:2502.20313*, 2025.

[18] D. P. Kingma, M. Welling, et al. Auto-encoding variational bayes, 2013.

[19] D. Lee, C. Kim, S. Kim, M. Cho, and W.-S. Han. Autoregressive image generation using residual quantization. In *CVPR*, 2022.

[20] T. Li, Y. Tian, H. Li, M. Deng, and K. He. Autoregressive image generation without vector quantization. *NeurIPS*, 2024.

[21] X. Li, K. Qiu, H. Chen, J. Kuen, J. Gu, B. Raj, and Z. Lin. Imagefolder: Autoregressive image generation with folded tokens. *arXiv preprint arXiv:2410.01756*, 2024.

[22] Y. Liu, L. Qu, H. Zhang, X. Wang, Y. Jiang, Y. Gao, H. Ye, X. Li, S. Wang, D. K. Du, et al. Detailflow: 1d coarse-to-fine autoregressive image generation via next-detail prediction. *arXiv preprint arXiv:2505.21473*, 2025.

[23] N. Ma, M. Goldstein, M. S. Albergo, N. M. Boffi, E. Vanden-Eijnden, and S. Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In *ECCV*, 2024.

[24] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[25] W. Peebles and S. Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

[26] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[27] A. Razavi, A. Van den Oord, and O. Vinyals. Generating diverse high-fidelity images with vq-vae-2. *NeurIPS*, 2019.

[28] S. Ren, Q. Yu, J. He, X. Shen, A. Yuille, and L.-C. Chen. Flowar: Scale-wise autoregressive image generation meets flow matching. *arXiv preprint arXiv:2412.15205*, 2024.

[29] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[30] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[31] P. Sun, Y. Jiang, S. Chen, S. Zhang, B. Peng, P. Luo, and Z. Yuan. Autoregressive model beats diffusion: Llama for scalable image generation. *arXiv preprint arXiv:2406.06525*, 2024.

[32] K. Tian, Y. Jiang, Z. Yuan, B. Peng, and L. Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *NeurIPS*, 2024.

[33] A. Van Den Oord, O. Vinyals, et al. Neural discrete representation learning. *NeurIPS*, 2017.

[34] X. Wang, X. Zhang, Z. Luo, Q. Sun, Y. Cui, J. Wang, F. Zhang, Y. Wang, Z. Li, Q. Yu, et al. Emu3: Next-token prediction is all you need. *arXiv preprint arXiv:2409.18869*, 2024.

[35] Y. Wang, Z. Wang, Z. Wu, Q. Tao, K. Liao, and C. C. Loy. Next visual granularity generation. *arXiv preprint arXiv:2508.12811*, 2025.

[36] H. Yu, H. Luo, H. Yuan, Y. Rong, and F. Zhao. Frequency autoregressive image generation with continuous tokens. *arXiv preprint arXiv:2503.05305*, 2025.

[37] J. Yu, Y. Xu, J. Y. Koh, T. Luong, G. Baid, Z. Wang, V. Vasudevan, A. Ku, Y. Yang, B. K. Ayan, et al. Scaling autoregressive models for content-rich text-to-image generation. *arXiv preprint arXiv:2206.10789*, 2022.

[38] Q. Yu, M. Weber, X. Deng, X. Shen, D. Cremers, and L.-C. Chen. An image is worth 32 tokens for reconstruction and generation. *NeurIPS*, 2024.

[39] K. Zha, L. Yu, A. Fathi, D. A. Ross, C. Schmid, D. Katabi, and X. Gu. Language-guided image tokenization for generation. In *CVPR*, 2025.

[40] X. Zhang, C. Da, H. Yang, K. Gai, M. Lu, and Z. Ma. Restok: Learning hierarchical residuals in 1d visual tokenizers for autoregressive image generation. *arXiv preprint arXiv:2601.03955*, 2026.

[41] B. Zheng, N. Ma, S. Tong, and S. Xie. Diffusion transformers with representation autoencoders. *arXiv preprint arXiv:2510.11690*, 2025.

A Appendix

A.1 Additional implementation details of RefineTok

Unless otherwise specified, all RefineTok tokenizer variants share one training recipe. Tokenizers are trained on ImageNet at 256×256 on 8 H200 GPUs, using AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$) with a peak learning rate of 2e-4, a 5K-step linear warmup, and cosine decay. We use a batch size of 256 and train for 200 epochs; ablation models instead use a 36-token budget and 50 epochs.

The tokenizer objective combines pixel reconstruction, perceptual, adversarial, and semantic alignment losses, where $\lambda_1 = 1.0$, $\lambda_2 = 0.75$, and $\lambda_3 = 0.02$. Our discriminator training largely follows the RAE [41] decoder training recipe. We adopt a DINO-based discriminator [3] with a frozen DINO-8 backbone. Compared with conventional PatchGAN-style discriminators, we find that using frozen DINO features provides a more stable perceptual adversarial signal and reduces high-frequency adversarial artifacts during tokenizer training. Following this setup, both real and reconstructed images are downsampled to 224 as discriminator inputs. Unless otherwise specified, we use standard batch normalization in the discriminator head.

A.2 Additional implementation details of generator

Our DiT-style generators are trained mostly following the standard DiT recipe. Since RefineTok produces compact 1D latent tokens, we use a patch size of 1 so that every latent token directly serves as one input token of the diffusion transformer. To preserve the scale-wise structure of the latent sequence, we incorporate 2D rotary positional embeddings [30] to distinguish tokens from different scales. We optimize the generator with AdamW at a learning rate of 2e-4 and a global batch size of 2048. For the main results, we train the DiT generator for 800 epochs. For ablation studies, we use shorter training schedules of 80 epochs (shown in Table 2) and we train the generator for 200 epochs to obtain more data points in Figure 5.

Sampling uses a standard Euler ODE solver with 250 steps in all experiments. For classifier-free guidance, class labels are dropped with probability 0.1 during training, and the guidance scale is chosen by grid search on validation. Unless otherwise specified, the guidance scale is set to 1.3 for DiT models trained with RefineTok-72/108 and 1.4 for DiT models trained with RefineTok-36.

A.3 Frequency analysis of refinement updates

To examine what each refinement stage adds, we bicubically resize the reconstructions at consecutive scales $s-1$ and s to 256×256 and compute their difference $\Delta_s = \mathcal{U}(\hat{x}_s) - \mathcal{U}(\hat{x}_{s-1})$, where $\mathcal{U}$ denotes bicubic resizing. We then compute the 2D Fourier power spectrum of Δ_s, group frequencies by their radial frequency normalized by the maximum radius, and report the fraction of spectral energy (excluding the DC component) in the low-, mid-, and high-frequency bands, averaged over 100 validation images. As shown in Table 5, early updates are dominated by low-frequency changes, whereas later updates increasingly contain mid- and high-frequency content, consistent with the coarse-to-fine behavior in Figure 8.

Table 5: Spectral energy distribution of reconstruction updates between adjacent scales of RefineTok-108.

Transition	Low $[0, 0.25)$	Mid $[0.25, 0.5)$	High $[0.5, 1]$
$16 \to 24$	99.62%	0.27%	0.12%
$24 \to 32$	99.55%	0.31%	0.14%
$32 \to 48$	99.68%	0.23%	0.10%
$48 \to 64$	99.50%	0.40%	0.10%
$64 \to 96$	97.02%	2.85%	0.12%
$96 \to 128$	93.06%	6.69%	0.25%
$128 \to 192$	75.42%	22.91%	1.67%
$192 \to 256$	48.67%	37.84%	13.49%



Figure 7: Additional progressive generation trajectories of RefineTok-108 on ImageNet 256×256. Each row illustrates how one generated sample is progressively refined across scales, from coarse structure to fine details. All intermediate outputs are obtained from a single forward pass of the decoder, without independently decoding each target scale.

A.4 Choice of semantic teacher

We study the Semantic Anchor with different pretrained semantic representations, keeping all other settings unchanged. All variants are trained for 20 epochs for a controlled comparison. As shown in Table 6, all aligned variants outperform the variant without SA in both PSNR and rFID, showing that SA is not tied to DINOv2. DINOv2-Base yields the best rFID, while increasing the teacher capacity to DINOv2-Large brings no clear improvement.

Table 6: Semantic Anchor with different semantic teachers. All variants are trained for 20 epochs.

Semantic teacher	PSNR↑	rFID↓
None	19.83	3.77
CLIP-Base	20.21	3.18
SigLIP2-Base	20.02	3.31
DINOv2-Base	20.16	2.82
DINOv2-Large	20.19	2.93

A.5 Progressive generation trajectory

Figure 7 provides additional progressive generation trajectories of RefineTok-108. Across different categories, the generated samples exhibit a coherent coarse-to-fine refinement process: early stages establish the global layout and object structure, while later stages gradually enrich local textures and fine-grained details. These examples further illustrate that the progressive refinement behavior of RefineTok is stable across diverse samples, rather than being limited to a few selected cases. Moreover, all intermediate outputs are obtained from a single decoder forward pass, highlighting that RefineTok naturally exposes the full generation trajectory without independently decoding each target scale.

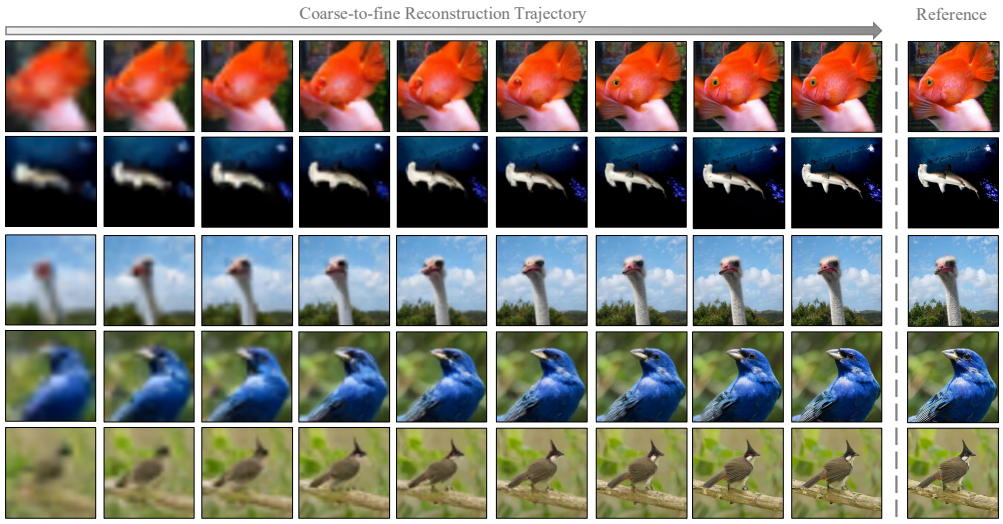


Figure 8: Progressive reconstruction trajectories of RefineTok-108 on ImageNet 256×256. Each row illustrates how the reconstruction is progressively refined across scales, from coarse structure to fine details, and the rightmost column shows the reference image. All intermediate outputs are obtained from a single forward pass of the decoder, without independently decoding each target scale.

A.6 Progressive reconstruction trajectory

Figure 8 shows progressive reconstruction trajectories of RefineTok-108. Across diverse examples, early stages already capture the coarse color distribution, object layout, and overall shape, while later stages progressively recover sharper boundaries, local textures, and fine-grained visual details. These results further support that RefineTok learns meaningful intermediate reconstruction states and maintains a coherent coarse-to-fine refinement process throughout decoding. Moreover, all intermediate reconstructions are produced in a single decoder forward pass, showing that RefineTok naturally exposes the full reconstruction trajectory without repeatedly decoding each scale independently.

A.7 More qualitative generation results

Figure 9 provides additional qualitative generation results for RefineTok under 72-token and 36-token budgets, respectively. The 72-token model consistently produces semantically meaningful samples with clear object structure and reasonably rich local details. Even with only 36 tokens, RefineTok still maintains coherent class semantics and plausible global layouts across a wide range of categories. These results further support that RefineTok provides a compact yet effective latent interface for downstream generation, and that its scale-wise progressive design remains robust even under very limited token budgets.



RefineTok-72



RefineTok-36

Figure 9: Additional qualitative class-conditional generation results on ImageNet 256×256 using RefineTok-36 and RefineTok-72 with a DiT-XL generator. RefineTok produces semantically meaningful images with clear object structure and rich fine-grained details under a compact 72-token budget. Despite the more compact 36-token budget, RefineTok still generates diverse and semantically coherent images.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Yes, our contribution is a progressive refinement decoder.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See the section of Discussion.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: No theory is included.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The code will be released upon completion of institutional review.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We conduct pre-training on ImageNet-1K and the variance is minor.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We train the models on 8 H200 GPUs which is described in paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have read and will follow the rule.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: There is no societal impact of the work.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper has no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have cited all the assets we use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: Our paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.