Locally Hierarchical Auto-Regressive Modeling for Image Generation

Tackgeun You[3,5] *
tackgeun.you@postech.ac.kr

Saehoon Kim[4]
shkim@kakaobrain.com

Chiheon Kim[4]
chiheon.kim@kakaobrain.com

Doyup Lee[4]
doyup.lee@kakaobrain.com

Bohyung Han[1,2,3]
bhhan@snu.ac.kr

[1]ECE, [2]IPAI, [3]AIIS, Seoul National University, Korea
[4]Kakao Brain, Korea
[5]CSE, POSTECH, Korea

Abstract

Auto-Regressive (AR) modeling on image generation provides a scalable solution to synthesize high-quality images by directly optimizing the data likelihood. For an efficient AR modeling of high-resolution images, many practical approaches decompose a learning process into the following two stages: (a) representing a high-resolution image by a sequence of codes, and (b) learning an AR model on this sequence, rather than on a sequence of pixels. This paper points out that determining the resolution of a single-scale code map triggers the trade-off between computational complexity and the fidelity of generated images. For example, high-fidelity image generation requires higher-resolution codes. But it needs exhaustive computation. To mitigate this trade-off, we propose a locally hierarchical auto-regressive model with different resolutions of codes. Our model represents an image by a pyramid of codes— coarse top code for high-level structure and fine bottom code for remaining details— via Hierarchical-Quantized Variational AutoEncoder (HQ-VAE) in the first stage. Hierarchical-Quantized Transformer (HQ-Transformer) in the second stage models a sequence of local hierarchies, which consist of a single top code and corresponding bottom codes. On class-conditional and text-conditional generation benchmarks, our model shows comparable performance to previous AR models in terms of the fidelity of generated images with lighter computational budgets.

1 Introduction

Deep generative models [11, 15, 35] have become the fundamental basis for synthesizing diverse images of high-resolution from various forms of input supervision, including class conditions [21, 35], text prompts [26, 27], and segmentation maps [9] to name a few. Most approaches in this paradigm could be roughly categorized into two types depending on whether they explicitly maximize the data likelihood or not. GANs implicitly estimate the underlying data distribution to generate novel and crispy samples, but it typically suffers from the lack of diversity [11]. In contrast, it is widely accepted that likelihood-based models are capable of generating diverse samples, compared to GANs, since they aim to cover all modes of data distribution [28]. In this paper, we focus on auto-regressive models among many likelihood-based approaches since the scalability of auto-regressive models is validated in several works [25, 27].

*This work was done during an internship at Kakao Brain.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Figure 1: Illustration on code-level disentanglement from HQ-Transformer (L) trained on ImageNet, where the first row shows samples synthesized only from the top codes, the second row from the bottom codes only, and the last row shows our final generation samples.

Auto-Regressive (AR) models factorize the data likelihood into a product of conditionals in pre-defined generation order. Besides the mode-coverage property of AR models, they require only minimal changes to architecture and optimization for being applied to various data modalities, such as audio, natural language, and images [24, 34, 35]. However, regardless of the modality, training AR models on high-dimensional samples is a very computationally demanding task, since each conditional needs to process many past observations. In the context of image generation, VQ-VAE [36] introduces a two-stage learning process, where stage 1 represents an image by a discrete code map, and stage 2 constructs AR priors [4, 35] on discrete sequences much shorter than the image. To make this two-stage framework effective, the codes need to have sufficient information to reconstruct the original image from the discrete sequence, while the sequence length should be reasonably short to train AR models.

In this study, we postulate that leveraging the multi-scale structure of visual images is an efficient way to represent an image as discrete codes for AR modeling. Suppose that we have multi-level codes obtained from stage 1, where the high-level code contains the overall structure and the low-level one encodes the detailed information. It now becomes possible to treat such codes differently, so an efficient solution could be presented in stage 2. This is because given the codes of low resolution, a light AR model may be enough to generate the codes of high resolution.

In order to achieve this goal, we propose locally hierarchical auto-regressive modeling extending two-stage AR modeling. We first extend VQ-VAE to produce a pyramid of two levels: the top and bottom codes, where the top contains the high-level information and the bottom encode the low-level details. Using these hierarchical discrete representations, an efficient AR model could be developed to generate a novel sample in a locally hierarchical manner. Specifically, the top codes are sequentially generated in a pre-defined order, and several bottom codes associated with a top code are decoded in parallel. The main contributions of the proposed model are summarized below:

- We propose HQ-VAE (Hierarchical-Quantized Variational Auto-Encoder) which encodes an image into a pyramid of two-level codes—coarse top codes and fine bottom codes— with hierarchical vector quantization. To fully utilize two-level codes, we enforce a disentanglement where coarse top code captures the high-level structure and fine bottom code to complement the remaining low-level details for the bottom code.
- We propose HQ-Transformer (Hierarchical-Quantized Transformer) built upon an assumption in local hierarchy that there exist several bottom codes conditionally independent given a top code. By predicting a sequence of local hierarchies with pre-defined order, we predict fine resolution codes with light computational budget.
- Our model achieves competitive performance on class- and text-conditional image generation, while observing high throughput in sample generation, compared to various types of generative models.

2 Related Works

In this section, we highlight the relevance and difference between previous approaches and ours.

Discrete representation for autoregressive models Learning AR priors on discrete representations have been proven to be effective in language modeling [24, 25]. In the case of continuous samples like images, we need to encode sufficient information of examples in the codes, while the sequence length should be short to train AR models. VQ-VAE pioneers a two-stage image generation process, where stage 1 represents an image by a set of discrete codes, and stage 2 constructs AR priors [4, 35] on discrete sequences. Many approaches to reducing the sequence length have been proposed to improve the stage 1 architecture [17, 38] or training objectives [9]. VQ-GAN [9] shortens the codes by using the perceptual and adversarial losses to encode the semantic information in the codes as much as possible. However, we observe that these approaches do not consider the multi-scale nature of an image, meaning that the receptive field of every code has the same size.

Hierarchical representation for autoregressive models To resolve the limitation of these single-scale discrete representation learning methods, many approaches to constructing hierarchical codes [10, 22, 28] have been proposed. VQ-VAE 2 [28] and HAM [10] make use of the hierarchical relationship between the codes to generate a novel image in a coarse-to-fine manner. RQ-Transformer [18] further reduces the sequence length by converting an image into a stack of code maps. This is constructed by iteratively quantizing the residual vector between the latent feature and code embedding, effectively reducing the latent feature map resolution without degrading the reconstruction quality. Hierarchical VQ-VAE [22] simply encodes the structural and textual information in different types of codes by training two separate VQ-VAEs, where the first is trained on the original images and the second on the residual between the original and reconstructed image. This process can disentangle the information in the code levels, but how to train AR models based on this set of codes is not studied. Constructed hierarchical codes can control the trade-off over code resolutions effectively. VQ-VAE2 [28] assigns a heavier model for coarse top code and a lighter model for fine bottom codes to trade-off the spatial resolution. RQ-Transformer with a stack of code maps from RQ-Transformer assigns a deep transformer for dealing with spatial resolution and a shallow transformer for depth. Our approaches also construct and exploits a pyramid of code map to effectively trade-off code resolution inspired from multi-scale nature.

Diffusion models Diffusion models [12, 15, 29] iteratively denoise a sample from the white noise by seeing the contexts in a bidirectional manner. Most diffusion models have been implemented in a U-Net [30] style architecture, meaning that the multi-scale feature of an image is well exploited. However, they suffer from low throughput in the generation, since the intermediate computation results between the denoising steps are not reusable, unlike AR models.

3 Locally Hierarchical Auto-Regressive Models

This section describes the details of the proposed architecture for image generation. Our approach exploits two levels of codes—top and bottom codes—for the representation of an image contrary to existing two-stage approaches based on AR models. The proposed architecture encourages the top code to model the overall structure of an image while facilitating the bottom code to complement the remaining details of the image. To this end, we propose Hierarchical-Quantized Variational AutoEncoder (HQ-VAE) for the image vector quantization in the first stage followed by HQ-Transformer to learn the prior over a sequence of the top and bottom codes in the second stage. Note that HQ-Transformer achieves better perceptual quality and sampling speed using the underlying hierarchical structure. Figure 2 illustrates the two stage procedure of our AR model.

3.1 Stage 1: Hierarchical-Quantized VAE (HQ-VAE)

We present the architecture and the optimization method of the proposed HQ-VAE, for which we first review the vector quantization technique and then discuss the hierarchical quantization module.

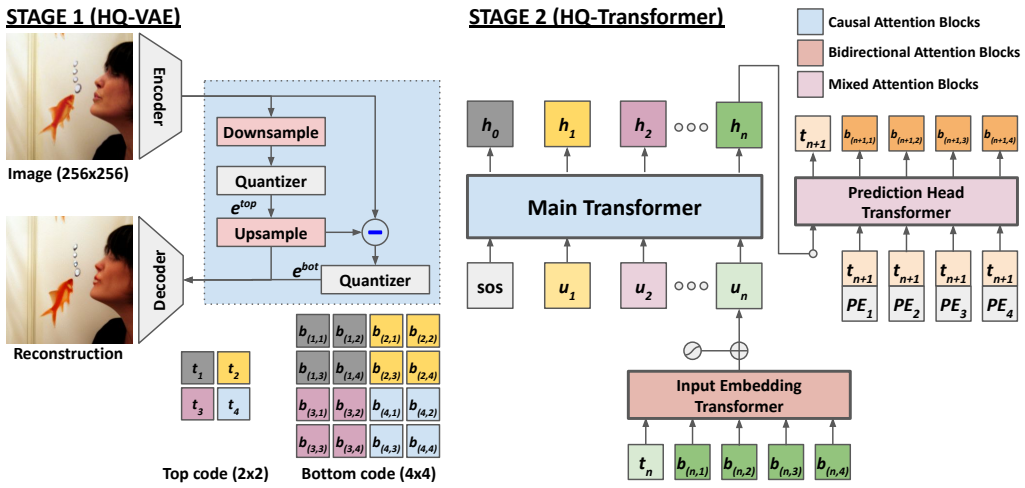


Figure 2: Conceptual illustration on the overall architecture: (a) the left one presents the components of HQ-VAE, where the residual and resampling operations enable to encode different information levels in the top and bottom codes, and (b) the right one shows the components of HQ-Transforms to train an AR prior on the top and bottom codes.

3.1.1 Notations for Vector Quantization

Vector quantization [20] maps a d-dimensional continuous vector into a code s. The mapping selects the nearest code embedding in a codebook, a collection of tuples $\{(s, \mathbf{e}(s))\}_{s \in n_{\mathcal{C}}}$, where $\mathbf{e}(s) \in \mathcal{R}^{d_e}$ denotes the embedding of code s and $n_{\mathcal{C}}$ is the size of codebook $\mathcal{C}$. Supposing that $\mathbf{z}_{ij}$ is a d-dimensional vector from the feature map $\mathbf{z} \in \mathbb{R}^{l \times l \times d}$ at the spatial location (i, j), the code s_{ij} and its embedding are given by

$$s_{ij} = \underset{1 \leq k \leq n_{\mathcal{C}}}{\operatorname{argmin}} ||\mathbf{z}_{ij} - \mathbf{e}(k)||, \quad \mathcal{C}(s_{ij}) = \mathbf{e}(s_{ij}), \tag{1}$$

where $\mathcal{C}(\cdot)$ returns the code embedding of an input code and $s_{ij} = Q(\mathbf{z}_{ij}; \mathcal{C})$ denotes the quantization process introduced for later use. Note that, since every element in a feature map has its own code, the spatial resolution of the code map, denoted by $\mathbf{s}$, is the same as that of the feature map.

3.1.2 Hierarchical Vector Quantization

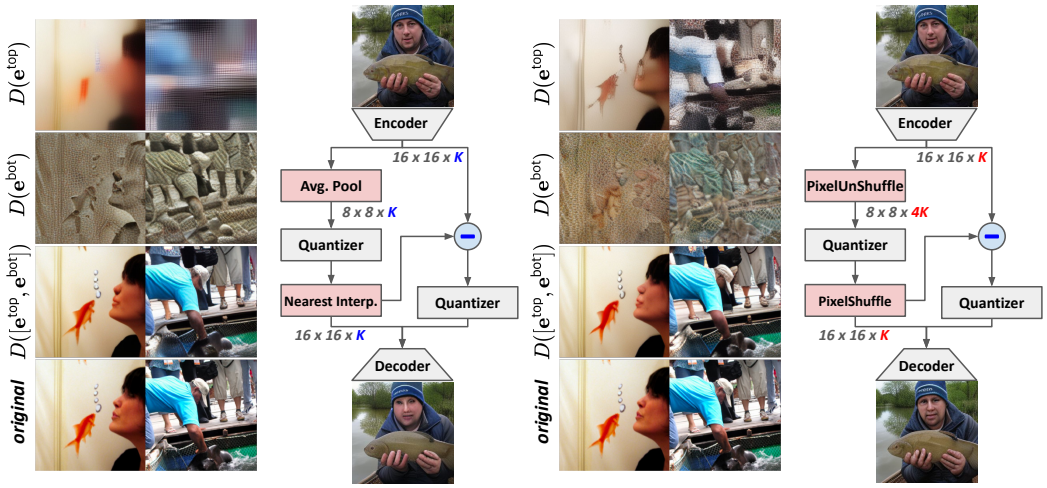
The objective of hierarchical vector quantization process is to encode different levels of information using the top and bottom codes. Intuitively, it is desirable for maximizing information in the codes for better fidelity of images. To maximize the contained information in two-level of the codes, we introduce a disentanglement scheme that encourages top codes to hold high-level information and bottom codes to encode remaining low-level details. This scheme results in maximum use of top and bottom codes. To accomplish this objective, our hierarchical quantization has two pathways for vector quantization process of top $\mathbf{t}$ and bottom code $\mathbf{b}$. In specific, Hierarchical vector quantization $\mathcal{H}(\cdot; r)$ transforms a feature map $\mathbf{z} \in \mathbb{R}^{rl \times rl \times d}$ into a tuple of two code maps $(\mathbf{t}, \mathbf{b})$, where $\mathbf{t} \in \mathcal{Z}^{l \times l}$ and $\mathbf{b} \in \mathcal{Z}^{rl \times rl}$ with an integer-valued scaling factor $r \geq 1$.

$$\mathbf{z}^{\text{top}} = \text{Downsample}(\mathbf{z}; r), \qquad \mathbf{t} = Q(\mathbf{z}^{\text{top}}; \mathcal{C}^{\text{top}}), \qquad \mathbf{e}^{\text{top}} = \mathcal{C}^{\text{top}}(\mathbf{t}), \tag{2}$$

where $\mathcal{C}^{\text{top}}$ denotes the codebook of top codes and $\text{Downsample}(\cdot)$ is the average pooling function to reduce the spatial resolution of a feature map. Based on $\mathbf{t}$, the bottom code and its embedding are derived by

$$\mathbf{z}^{\text{bot}} = \mathbf{z} - \text{Upsample}(\mathbf{e}^{\text{top}}; r), \quad \mathbf{b} = Q^{\text{bot}}(\mathbf{z}^{\text{bot}}; \mathcal{C}^{\text{bot}}), \quad \mathbf{e}^{\text{bot}} = \mathcal{C}^{\text{bot}}(\mathbf{b}), \tag{3}$$

where $\mathcal{C}^{\text{bot}}$ is the codebook of bottom codes and $\text{Upsample}(\cdot)$ is any upsampling operation to increase the spatial resolution of the feature map. Figure 2(left) illustrates this hierarchical quantization process.



(a) Average pooling and nearest interpolation

(b) Pixel-unshuffle and pixel-shuffle

Figure 3: Comparison of different implementations of resampling operations in our hierarchical vector quantization procedure. In Figure 3a, averaging pooling and nearest interpolation are used for resampling, where the overall structure is not clearly shown in the reconstructed image from the top codes only. In Figure 3b, instead of pooling operations, pixel-unshuffle and -shuffle operations are employed to encode more information in the top codes.

Our scheme first captures the high-level information of a feature map by quantizing its downsampled version based on the top codebook. Hence, the residual vector between the original feature map and the upsampled top code embedding map $\mathbf{z}^{\text{bot}}$ contains resampling and quantization errors. Figure 3 compares the reconstructed images from each of the top and bottom code embeddings, demonstrating the disentanglement enforced by the proposed hierarchical vector quantization.

We first instantiate resampling operations in hierarchical quantization by average pooling for downsampling and nearest-neighbor interpolation for upsampling. Despite the reasonable performance given by these naive resampling operations, we observe that the overall structure reconstructed by the top code embeddings is not strong, as shown in Figure 3a. To push more information into the top codes, we have to reduce the information loss caused by the downsampling operation as much as possible. A simple yet effective solution to this challenge is to employ pixel-unshuffle and -shuffle operations [33]. Given the feature map $\mathbf{z} \in \mathbb{R}^{rl \times rl \times d}$ and a scaling factor r, the pixel-unshuffle operation returns a tensor with the shape $l \times l \times r^2 d$ and maintains all information in the original feature map. To compute the residual, we shuffle the top code embedding and restore the original dimensionality of $\mathbf{z}$. Figure 3 clearly demonstrates that the top codes constructed by the pixel-shuffle contain the primary information of an input image more than the ones given by pooling operations.

We observe that non-learnable layers give a straightforward solution for resampling operations over naive learnable layers. A sophisticated design of additional loss functions for top codes is needed to avoid top code collapse because there is no incentive to encode the global information in the top code. However, the naive design of hierarchical quantization only relies on high-resolution bottom codes and fails to exploit low-resolution top codes due to no incentive for using top codes with smaller resolutions in vector quantization. Even if there exists an additional loss function, resampling operations with the learnable layers perform worse. We discuss the design of the loss function on the experiment section in detail.

3.1.3 HQ-VAE: Architecture & Optimization

Overall structure HQ-VAE consists of three parts: encoder $E(\cdot; f)$, decoder $D(\cdot)$, and hierarchical quantizer $\mathcal{H}(\cdot; r)$, as shown in Figure 2, where f is the initial downsampling factor and r is the additional scaling factor for the top code. Given an image $\mathbf{x} \in \mathbb{R}^{h \times w \times 3}$, an encoder $E(\cdot)$ extracts the feature map $\mathbf{z} \in \mathbb{R}^{\frac{h}{f} \times \frac{w}{f} \times d}$. Then, the hierarchical quantizer $\mathcal{H}(\cdot; r)$ is applied to the feature map,

providing the top and bottom codes $(\mathbf{t}, \mathbf{b})$ and their associated code embeddings: $\mathbf{e}^{\text{top}}$ and $\mathbf{e}^{\text{bot}}$. Now, the decoder $D(\cdot)$ takes the top and bottom code embeddings to reconstruct the input image.

Objective function To train three parts in HQ-VAE, our objective function is composed of the reconstruction loss introduced in VQ-VAE [36] as well as adversarial and perceptual losses to improve the fidelity of reconstructed images [9]. Here, we only present the reconstruction loss as below:

$$\mathcal{L}_{\text{rec}}(\mathbf{x}; E, D, \mathcal{H}) = \|\mathbf{x} - D(\mathbf{e}^{\text{top}}, \mathbf{e}^{\text{bot}})\|_2^2 \quad + \sum_{i \in \{\text{top,bot}\}} \left[\|\text{sg}[\mathbf{z}^i] - \mathbf{e}^i\|_2^2 + \beta \|\text{sg}[\mathbf{e}^i] - \mathbf{z}^i\|_2^2\right], \quad (4)$$

where $\text{sg}[\cdot]$ refers to a stop-gradient operation, and β is set to 0.25 as suggested in [36]. The first term measures L_2 distance between the reconstructed image and the original. The second and third terms enforce the code embedding close to the inputs of quantizers. Since we have two types of codes, we simply sum up the second and third terms of each type. To update the codebook, we make use of exponential moving average (EMA), because this accelerates the convergence [36].

Implementation details We adopt the same network structures for encoder and decoder as in VQ-GAN [9] with two minor modifications: (a) to increase throughput in training, we set the stride of the first convolution layer in the encoder and the last transposed convolution layer in the decoder as two, instead of one in the original implementation. (b) we replace the batch normalization [16] in a discriminator by group normalization [37] to achieve the consistent results regardless of a batch size.

3.2 Stage 2: Hierarchical-Quantized Transformer (HQ-Transformer)

In this section, we propose HQ-Transformer (see Figure 2) to autoregressively model the top and bottom codes by fully exploiting their spatial relationships obtained from the first stage.

3.2.1 Formulation

Our objective is to maximize the log-likelihood of a tuple of $\mathbf{t}$ and $\mathbf{b}$. Given the scaling factor r for the top, the resampling operation aligns a single top code t_i with r^2 bottom codes $b_{(i,k)}$, where $1 \leq k \leq r^2$ for every i. An example of top and bottom codes with $r = 2$ is shown in the bottom of Figure 2. We denote all bottom codes aligned with a single top code t_i as $\mathbf{b}_i = (b_{(i,1)}, b_{(i,r^2)})$. Given that the top code map shape is $l \times l$ and the sequence order of the top is a typical raster scan order, the likelihood function can be factorized below:

$$\mathbb{P}_\theta(\mathbf{t}, \mathbf{b}) = \mathbb{P}_\theta\left(t_1 \cdots t_{l^2}, b_{(1,1)} \cdots b_{(l^2, r^2)}\right) = \prod_{i=1}^{l^2} \mathbb{P}_\theta\left(t_i, \mathbf{b}_i | \mathbf{t}_{<i}, \mathbf{b}_{<i}\right), \quad (5)$$

where the notation $< i$ denotes a collection of j when $j < i$. The conditional can be further factorized into:

$$\mathbb{P}_\theta(t_i, \mathbf{b}_i | \mathbf{t}_{<i}, \mathbf{b}_{<i}) = \mathbb{P}_\theta(\mathbf{b}_i | t_i, \mathbf{t}_{<i}, \mathbf{b}_{<i}) \, \mathbb{P}_\theta(t_i | \mathbf{t}_{<i}, \mathbf{b}_{<i}), \quad (6)$$

where the last term can be maximized by predicting the current position given the previous top and bottom codes. For the first term, we observe that a reasonable assumption on the bottom codes leads to more efficient formulation.

Here, we assume that the bottom codes are conditionally independent given the associated top code, since bottom codes contain only detailed local information if the top code has global-level information, which is enforced by our stage 1 model, HQ-VAE. Therefore, we further expand the first term in Eq. 6 as

$$\mathbb{P}_\theta(\mathbf{b}_i | t_i, \mathbf{t}_{<i}, \mathbf{b}_{<i}) \triangleq \prod_{k=1}^{r^2} \mathbb{P}_\theta\left(b_{(i,k)} | t_i, \mathbf{t}_{<i}, \mathbf{b}_{<i}\right), \quad (7)$$

3.2.2 Architectural Details

Similar to GPT-like architectures [3, 25], HQ-Transformer consists of three sub-modules—input embedding, main transformer, and prediction head. To effectively aggregate the information from the top and bottom codes for the main transformer and making a prediction, we design the input embedding and prediction head by a Transformer instead of linear projection.

Input Embedding Transformer (IET) IET transforms a tuple of $(t_i, \mathbf{b}_i)$ into a single embedding vector $\mathbf{u}_i$, for all i. In order to do that, we first construct a local embedding by summing up the code embedding and positional encoding:

$$\begin{aligned} \mathbf{m}_i^{\text{top}} &= \mathbf{e}_i^{\text{top,in}} + \text{PE}^{\text{top}}(i) + \text{PE}^{\text{bot}}(0) \\ \mathbf{m}_{(i,k)}^{\text{bot}} &= \mathbf{e}_{(i,k)}^{\text{bot,in}} + \text{PE}^{\text{bot}}(k), \qquad \text{for } k = 1, \cdots, r^2, \end{aligned} \tag{8}$$

where $\mathbf{e}_i^{\text{top,in}}$ and $\mathbf{e}_{(i,k)}^{\text{bot,in}}$ mean the corresponding learnable code embedding of t_i and $b_{(i,k)}$ and $\text{PE}^{\text{top}}[\cdot]$ and $\text{PE}^{\text{bot}}[\cdot]$ denote learnable positional encodings for the top and bottom. Note that $\text{PE}^{\text{bot}}[0]$ is introduced to distinguish between the local embeddings from the top and bottom codes. Now, these local embeddings are refined by a bidirectional transformer:

$$\mathbf{u}_i^{\text{top}}, \mathbf{u}_{(i,1)}^{\text{bot}}, \cdots, \mathbf{u}_{(i,r^2)}^{\text{bot}} = \text{InputEmbedding}\left(\mathbf{m}_i^{\text{top}}, \mathbf{m}_{(i,1)}^{\text{bot}}, \cdots, \mathbf{m}_{(i,r^2)}^{\text{bot}}\right), \tag{9}$$

where the structure is the same with GPT-2 [25]. And we observe that it is sufficient to employ a single GPT-2 block (composed of a self-attention and MLP block). Then, the final input embedding $\mathbf{u}_i$ for the main Transformer block is simply averaging all updated local embeddings:

$$\mathbf{u}_i = \text{Average}(\mathbf{u}_i^{\text{top}}, \mathbf{u}_{(i,1)}^{\text{bot}}, \cdots, \mathbf{u}_{(i,r^2)}^{\text{bot}}). \tag{10}$$

Main Transformer This module updates the outputs of IET by a stack of N_{main} causal self-attention and MLP blocks:

$$\mathbf{h}_0, \mathbf{h}_1, \cdots, \mathbf{h}_{l^2} = \text{MainTransformer}(\text{SOS}, \mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_{l^2}), \tag{11}$$

where the sequence length is the number of top codes and the start-of-sentence token (SOS) is located at the first of input embedding. Here, we use the same structure of GPT-2.

Prediction Head Transformer (PHT) This module predicts the top and bottom codes from the hidden representation of the main transformer, where the bottom codes at the next position are generated in parallel based on their associated top code and a hidden representation from the main Transformer. In specific, we first define the input sequence of PHT as below:

$$\mathbf{v}_i^{(k)} = \begin{cases} \mathbf{h}_i + \text{PE}^{\text{head}}(k) & \text{if} \quad k = 0, \\ \mathbf{e}_i^{\text{top,head}} + \text{PE}^{\text{head}}(k) & \text{if} \quad 1 \le k \le r^2, \end{cases} \tag{12}$$

where $\mathbf{e}_i^{\text{top,head}}$ means the corresponding learnable code embedding of t_i. This sequence is then updated to produce the conditional probabilities of top and bottom codes:

$$p_i^{(0)}, p_i^{(1)}, \cdots, p_i^{(r^2)} = \text{TransformerHead}(\mathbf{v}_i^{(0)}, \mathbf{v}_i^{(1)}, \cdots, \mathbf{v}_i^{(r^2)}), \tag{13}$$

where each probability is defined as

$$p_i^{(k)} = \begin{cases} \mathbb{P}\left(t_i | \mathbf{t}_{<i}, \mathbf{b}_{(<i)}\right) & \text{if} \quad k = 0, \\ \mathbb{P}\left(b_{(i,k)} | t_i, \mathbf{t}_{<i}, \mathbf{b}_{<i}\right) & \text{if} \quad 1 \le k \le r^2. \end{cases} \tag{14}$$

Here, we use the self-attention mask pattern as a mixture of causal and bidirectional mask—the causal attention from the top to bottom codes and the bidirectional attention between the bottoms. Compared to full-autoregressively modeling on the bottoms, this parallel decoding achieves faster sampling and even better generation quality. We will discuss the effect of this parallel decoding in the experiments section.

3.2.3 Computational complexity of HQ-Transformer

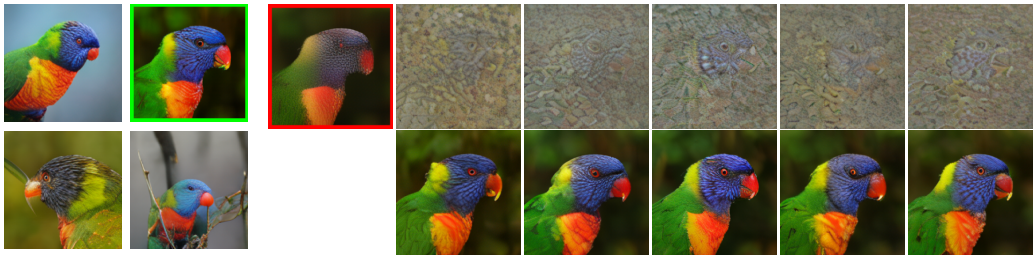
Suppose that we have the top code $\mathbf{t} \in \mathcal{Z}^{l \times l}$ and the bottom code $\mathbf{b} \in \mathcal{Z}^{rl \times rl}$, where r is a scaling factor for the top code quantizer. We denote $L_{\text{top}} = l^2$ as the length of the top codes and $L_{\text{local}} = 1 + r^2$ as the number a single top code and all bottom codes associated with the top code. The time complexities of input embedding, main, and prediction head transformer are then $O(N_{\text{in}} L_{\text{top}} L_{\text{local}}^2)$, $O(N_{\text{main}} L_{\text{top}}^2)$ and $O(N_{\text{head}} L_{\text{top}} L_{\text{local}}^2)$, where N_{in}, N_{main}, and N_{head} are the number of attention blocks. And, $O(N_{\text{main}} L_{\text{top}}^2)$ dominates the other terms, since N_{main} is much larger than N_{in} and N_{head}. Suppose that the code map resolution of VQ-GAN is equal to the bottom code resolution of HQ-Transformer. Computational complexity of HQ-Transformer is mainly $O(N_{\text{main}} L_{\text{top}}^2)$, which is much lower than the complexity of VQ-GAN $O(N_{\text{main}} L_{\text{top}}^2 r^4)$.

Table 1: Class-conditional image generation performance on ImageNet in terms of FID, precision, recall, and throughput. The values in parenthesis are acquired by rejection sampling.

Models	Parameter size	Code map resolution	FID ↓	Precision ↑	Recall ↑	Throughput ↑ # images/sec
ADM [7]	554 M	-	10.94 (4.59)	0.69 (0.82)	0.63 (0.52)	0.12 (0.07)
ImageBART [8]	3.5 B	16×16	21.19 (7.44)	-	-	-
VQ-Diffusion [12]	370 M	32×32	11.89 (5.32)	-	-	2.66 (0.13)
LDM-4 [29]	400 M	64×64	10.56 (3.60)	0.71 (0.87)	0.62 (0.48)	0.70 (0.30)
MaskGIT [1]	228 M	16×16	6.18 (4.02)	0.80	0.51	11.04 (0.55)
VQ-VAE 2 [28]	13.5 B	$32 \times 32 + 64 \times 64$	~ 31 (~ 10)	0.36	0.57	-
ViT-VQGAN [38]	1.6 B	32×32	4.17 (3.04)	-	-	-
VQ-GAN [9]	1.4 B	16×16	15.78 (5.20)	-	-	6.76 (0.34)
RQ-Transformer [18]	1.4 B	$8 \times 8 \times 4$	8.71 (3.89)	0.71	0.58	37.03 (1.85)
RQ-Transformer [18]	3.8 B	$8 \times 8 \times 4$	7.55 (3.80)	0.73 (0.82)	0.58 (0.50)	23.22 (2.90)
HQ-Transformer (S)	530 M	$8 \times 8 + 16 \times 16$	9.36 (4.98)	0.69 (0.78)	0.55 (0.51)	78.87 (19.72)
HQ-Transformer (M)	870 M	$8 \times 8 + 16 \times 16$	8.46 (4.47)	0.71 (0.79)	0.56 (0.52)	70.00 (17.50)
HQ-Transformer (L)	1.4 B	$8 \times 8 + 16 \times 16$	7.15 (4.35)	0.73 (0.81)	0.55 (0.50)	49.44 (12.36)

Table 2: Text-to-image generation performance on CC3M validation set.

Models	Parameter size	Code map resolution	FID ↓	CLIP-score ↑	Throughput ↑
ImageBART [8]	2.8 B	16×16	22.61	0.23	1.75
LDM [29]	645 M	64×64	17.01	0.24	3.9
VQ-GAN [9]	1.5 B	16×16	28.86	0.20	2.69
RQ-Transformer [18]	654 M	$8 \times 8 \times 4$	12.33	0.26	46.47
HQ-Transformer (S)	579 M	$8 \times 8 + 16 \times 16$	12.86	0.26	54.88



(a) Generated samples (b) Top code-conditional image generation

Figure 4: The process of top code-conditional image generation. Generated images with ImageNet class *lorikeet* (90) are shown in Figure 4a. We extract the top codes from the green bordered image. The red bordered image visualizes an image reconstructed from the extracted top codes only. Given the top codes, HQ-Transformer samples bottom codes visualized at the first row in Figure 4b and generates realistic images with various details at the second row in Figure 4b.

4 Experiments

In the experiment section, we evaluate our models on class- and text-conditional image generation tasks. We observe that HQ-Transformer achieves competitive performance to other types of generation models, while allowing high throughput in sample generation. We also present various ablation studies on our architecture to support the rationale of our choice.

4.1 Conditional Image Generation

Settings We train our models on 1.2M samples from ImageNet [6] trainset for a class-conditional image generation task. In case of a text-conditional task, we use 15M image-text pairs constructed by Conceptual Caption (CC) [32] and Conceptual-12M [2]. In training phase of both cases, we resize the short axis of an image to 256 and randomly crop the image to acquire an image of 256×256 resolution. In testing, we also resize the short axis to 256 and crop the image at the center.

We setup HQ-VAE by $f = 8$ and $r = 2$ to produce the top and bottom codes of 8×8 and 16×16 resolution, respectively. We instantiate three types of HQ-Transformer: (a) $L_{\text{main}} = 12$, and $L_{\text{head}} = 4$ for the smallest model, (b) $L_{\text{main}} = 24$ and $L_{\text{head}} = 4$ for the middle-size model, and (c) $L_{\text{main}} = 42$,

Table 3: Comparison of image reconstruction quality on ImageNet validation set. RQ-VAE* is referred to as our modified version by adopting the same architecture used in HQ-VAE.

Model	Code map resolution	rFID ↓	Codebook size	Effective code usage
DALL-E [27]	32×32	32.01	8192	-
AugVAE-SL [17]	32×32	3.28	8192	-
ViT-VQGAN [38]	32×32	1.28	8192	-
VQ-GAN [9]	16×16	4.90	16384	973
RQ-VAE [18]	$8 \times 8 \times 4$	3.20	16384	16353
RQ-VAE* [18]	$8 \times 8 \times 5$	3.37	16384	16322
HQ-VAE ($f = 16, r = 2$)	$8 \times 8 + 16 \times 16$	2.61	8192 + 8192	8119 + 8192
HQ-VAE ($f = 8, r = 2$)	$16 \times 16 + 32 \times 32$	0.57	8192 + 8192	8192 + 8192

Table 4: Ablation study on HQ-VAE.

Resampling op.	Epochs	rFID ↓
Conv2 + Perceptual loss	15	3.72
Avg. Pool.	15	3.67
PixelShuffle	15	3.14
PixelShuffle	50	2.61

Table 5: Ablation study on HQ-Transformer

Decoding policy	Label type	FID ↓
VQ-VAE2 style	One-hot	23.25
Causal	One-hot	11.33
Parallel		10.01
Parallel	Soft-label ($\tau = 1$)	9.36

and $L_{\text{head}} = 6$ for our largest model. In all cases, $L_{\text{in}} = 1$ and other parameters related to the network size are the same. We describe more details on the network architecture in the supplementary material. For evaluation measure, we use Fréchet Inception Distance (FID) [14], precision & recall [31] in class-conditional generation, and FID and CLIP-score [23] in a text-conditional generation task. We measure throughput in sample generation on a single Tesla A100 in all cases.

Class-conditional image generation Table 1 shows that our largest model, HQ-Transformer (L), achieves the best FID 7.15 compared to two-stage AR models on the similar sequence length, VQ-GAN (16×16) and RQ-Transformer ($8 \times 8 \times 4$). We observe that HQ-Transformers report the highest throughput in sample generation than that of other models, since ours generate r^2 bottom codes in parallel given the associated top code. Compared to ADM, our smallest model (S) can generate samples of similar quality, while achieving 650× larger throughput. We also conduct rejection sampling by using ResNet-101 [13] with acceptance rate 0.25 for all our models, and the results are marked in the parenthesis in Table 1. Here, our models show competitive performance to the other methods. Figure 4 demonstrates top code-conditional image generation to show the disentanglement of top and bottom codes. Contrary to typical generative models, HQ-Transformer restricts the diversity of generated images within details by sampling only bottom codes from HQ-Transformer with fixed top code.

Text-conditional image generation Table 2 reports FID and CLIP-score of several models measured on CC3M validation split, and throughput in sample generation. This shows that our model is comparable to RQ-Transformer in terms of FID (12.86 vs. 12.33) and CLIP-score (0.26 vs. 0.26), while achieving 18% higher throughput than that of RQ-Transformer. This result shows that our model behaves similarly regardless of data modality.

4.2 Detailed Experiments on HQ-VAE and HQ-Transformer

Comparison with the other stage 1 methods Table 3 compares two types of HQ-VAE, ($f = 16, r = 2$) and ($f = 8, r = 2$), with the existing stage 1 models, where we measure FID between the original and reconstructed image, referred to as rFID. We also measure the number of codes activated in reconstruction, and report it in the last column. rFID from HQ-VAE ($8 \times 8 + 16 \times 16$) is better than the ones from baselines with the similar sequence length: RQ-VAE ($8 \times 8 \times 4$) and RQ-VAE ($8 \times 8 \times 5$). In addition, HQ-VAE ($16 \times 16 + 32 \times 32$) achieves better than that of ViT-VQGAN (32×32), while the efficient Transformer can be implemented on the hierarchical code maps, unlike ViT-VQGAN.

Ablation study on HQ-VAE Table 4 supports the rationale of our choice, showing that resampling with pixel-shuffle and -unshuffle outperforms the one with average pooling and nearest interpolation by 0.53 in FID, since our choice prevents from loosing information when downsampling. We also observe that longer training boosts rFID performance from 3.14 to 2.61. We instantiate learnable

Table 6: Comparison of two- and three-level HQ-VAE and HQ-Transformer. All HQ-VAEs and HQ-Transformers are trained for 15 and 100 epochs correspondingly for fair comparison.

Code map resolution	rFID ↓	rFID (top) ↓	Effective code usage	FID	Precision	Recall
4x4 + 8x8	18.65	113.21	7524 + 8192	37.59	0.49	0.38
4x4 + 8x8 + 16x16	2.48	186.75	1954 + 7763 + 8192	15.66	0.61	0.54
8x8 + 16x16	3.14	83.45	8008 + 8192	13.59	0.67	0.52
8x8 + 16x16 + 32x32	0.65	135.29	2656 + 8192 + 8192	11.10	0.64	0.58

resampling operations with convolution and strided convolution layers with kernel size 2 and stride 2. To avoid top code collapse, we apply additional perceptual loss on top code only reconstructed images. HQ-VAE with perceptual loss achieves reconstruction FID 3.72 on ImageNet validation split. This performance is worse than HQ-VAE with average pooling and pixel-shuffle reported in Table 4. Refer the detailed analysis on the appendix.

Ablation study on HQ-Transformer Table 5 compares decoding policies and highlights proposed parallel decoding. VQ-VAE 2 [28] style decoding generates top codes in advance and bottom codes conditioned on top codes strictly hierarchical order. Two standard transformers are utilized in this policy for top code generation and top conditioned bottom code generation. Three types of locally hierarchical policies are listed. Causal decoding from RQ-Transformer [18] generates top code and related local bottom codes in a sequential manner. Parallel decoding indicates our procedure; the top code is first generated and related r^2 bottom codes are decoded in parallel. Locally hierarchical decoding policies outperform strictly hierarchical decoding policy because the full visual information can be recovered by combining both top and bottom codes. We observe that our parallel generation scheme achieves better FID than a locally causal decoding, since error accumulation during sampling is alleviated by our parallel decoding approach. We also validate the effects of soft-label as an alternative to one-hot label in cross-entropy loss function for training HQ-Transformer as proposed in RQ-Transformer [18]. In our case, using soft-labels with temperature $\tau = 1$ improves the performance similarly observed in [18].

Multi-level extensions Multi-level extensions of our approach have the potential to improve the quality of image generation. Compared to the two-level architectures with the same top code resolutions, three-level architectures outperform their two-level counterparts. Adding 16×16 bottom codes on $(4 \times 4 + 8 \times 8)$ significantly boosts FID performance from 37.59 to 15.66. The model with fine three-level codes $(8 \times 8 + 16 \times 16 + 32 \times 32)$ also outperforms its two-level counterparts. However, we could not achieve the best performance using the three-level architecture with code map resolution $(8 \times 8 + 16 \times 16 + 32 \times 32)$. Contrary to two-level architecture, longer training of three-level HQ-VAE does not improve the performance of both HQ-VAE and HQ-Transformer. We conjecture that it is because of lower usage of top codes in three-level HQ-VAE as shown in Table 6.

5 Conclusion

We introduce a two-stage auto-regressive model (HQ-VAE and HQ-Transformer) to generate a novel sample in a locally hierarchical manner extending typical two-stage auto-regressive models. HQ-VAE represents an image by a top and bottom codes with multiple resolutions, encoding the overall high-level structure of an image in the top codes of low-level and detailed information in the bottom codes of high resolution. HQ-Transformer models these top and bottom codes efficiently and generates novel images of high quality. Experiments confirm that our approach is capable of generating high-quality samples compared to recent architectures, while achieving high throughput in sample generation. A multi-level extension of our approaches shows potential advantages for higher-resolution image generation. We expect that tackling lower top code usage in multi-level HQ-VAE leads to better generation performance with faster sampling speed.

Acknowledgement This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) [No.2022-0-00959, (Part 2) Few-Shot Learning of Causal Inference in Vision and Language for Decision Making; No.2022R1A5A708390811, Trustworthy Artificial Intelligence]

References

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

[2] S. Changpinyo, P. Sharma, N. Ding, and R. Soricut. Conceptual 12M: Pushing web-scale image-text pre-training to recognize long-tail visual concepts. In *CVPR*, 2021.

[3] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever. Generative pretraining from pixels. In *ICML*, 2020.

[4] X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel. Pixelsnail: An improved autoregressive generative model. In *ICML*, 2017.

[5] C. Christopoulos, A. Skodras, and T. Ebrahimi. The jpeg2000 still image coding system: an overview. *IEEE Transactions on Consumer Electronics*, 2000.

[6] 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.

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

[8] P. Esser, R. Rombach, A. Blattmann, and B. Ommer. Imagebart: Bidirectional context with multinomial diffusion for autoregressive image synthesis. In *NeurIPS*, 2021.

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

[10] J. D. Fauw, S. Dieleman, and K. Simonyan. Hierarchical autoregressive image models with auxiliary decoders. *arXiv preprint arXiv:1903.04933*, 2019.

[11] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. In *NeurIPS*, 2014.

[12] S. Gu, D. Chen, J. Bao, F. Wen, B. Zhang, D. Chen, L. Yuan, and B. Guo. Vector quantized diffusion model for text-to-image synthesis. In *CVPR*, 2022.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[14] 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. In *NeurIPS*, 2017.

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

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[17] T. Kim, G. Song, S. Lee, S. Kim, Y. Seo, S. Lee, S. H. Kim, H. Lee, and K. Bae. L-verse: Bidirectional generation between image and text. In *CVPR*, 2022.

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

[19] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[20] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability, 1967: 281-297*, 1967.

[21] M. Mirza and S. Osindero. Conditional generative adversarial nets. In *arXiv*, 2014.

[22] J. Peng, D. Liu, S. Xu, and H. Li. Generating diverse structure for image inpainting with hierarchical vq-vae. In *CVPR*, 2021.

[23] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

[24] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2018.

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

[27] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *ICML*, 2021.

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

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

[30] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[31] M. S. M. Sajjadi, O. Bachem, M. Lucic, O. Bousquet, and S. Gelly. Assessing generative models via precision and recall. In *NeurIPS*, 2018.

[32] P. Sharma, N. Ding, S. Goodman, and R. Soricut. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *ACL*, 2018.

[33] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *CVPR*, 2016.

[34] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[35] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders. In *NeurIPS*, 2016.
[36] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *NeurIPS*, 2017.
[37] Y. Wu and K. He. Group normalization. In *ECCV*, 2018.
[38] J. Yu, X. Li, J. Y. Koh, H. Zhang, R. Pang, J. Qin, A. Ku, Y. Xu, J. Baldridge, and Y. Wu. Vector-quantized image modeling with improved vqgan. In *ICLR*, 2022.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We discuss the limitation of our models on the supplementary material.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We discuss it in the supplementary material
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We report in both the paper and the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We report in both the paper and the supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] The evaluation protocol for conditional generative models does not report the error bars. Instead, we fix the random seed and make experiments reproducible.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We report in both the paper and the supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cite and report in the supplementary material.
   (b) Did you mention the license of the assets? [Yes] We report in the supplementary material.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Implementation details

A.1 HQ-VAE

We adopt and modify the encoder and the decoder architecture from VQ-GAN [9]. Contrary to VQ-GAN, we set the stride of the first convolution layer in the encoder and the last transposed convolution layer in the decoder as two, instead of one in the original implementation as mentioned in the main paper. The encoder takes an image $\mathbf{x} \in \mathbb{R}^{256\times256\times3}$ and returns encoded feature map $\mathbf{z} \in \mathbb{R}^{\frac{256}{f}\times\frac{256}{f}\times256}$. We adopt HQ-VAE equipped with pixel-unshuffle and -shuffle for resampling. HQ-VAE with $(f = 16, r = 2)$ has a code map resolution $8 \times 8 + 16 \times 16$ and $(f = 8, r = 2)$ has code map resolution $16 \times 16 + 32 \times 32$. We use HQ-VAE $(f = 16, r = 2)$ for all experiments, which gives concise visual codes for efficient training of HQ-Transformer.

We train HQ-VAE $(f = 16, r = 2)$ for 50 epochs on ImageNet training split. We use Adam optimizer with $(\beta_1, \beta_2) = (0.9, 0.5)$ and batch size 128. We set a learning rate 4×10^{-5} with warm-up 0.5 epochs starting from zero learning rate. Loss weights for reconstruction, perceptual, codebook, and discriminator are set to 1.0, 1.0, 1.0, and 0.75 respectively.

A.2 HQ-Transformer

Overall hyperparameters for HQ-transformer are shown in Table A1. d_e and n_e denote the embedding dimension and the number of heads for the transformer.

We use the same notations for parameters in the main paper. Input embeddings for the main transformer start with start-of-sentence (SOS) tokens. Class-conditional image generation uses class-conditional learnable embedding as a single SOS token. Text-conditional image generation uses embeddings of a sequence of text tokenized by a byte pair encoding [**? ?**]. The length of text tokens is at most 64 text tokens.

We train HQ-Transformer 100 epochs on ImageNet training split and 20 epochs on CC-15M, aggregated training splits of CC-12M and CC-3M. Training HQ-transformer (S), (M), and (L) on ImageNet takes 35, 45, and 75 minutes per epoch with four NVIDIA Tesla A100. Training HQ-transformer (S) on CC-15M takes 452 minutes per epoch with four NVIDIA Tesla A100. We use HQ-VAE trained on ImageNet for experiments on class- and text-conditional image generation. We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, batch size 512 and weight decay 1×10^{-4}. We adopt two policies for learning rate schedules, warm-up, and cosine annealing. At the first epoch, learning rate is warm-up gradually from $\text{lr}_{\text{init}} = 1 \times 10^{-5}$ to lr_{peak}. In the remaining training epochs, we use cosine annealing from peak learning rate lr_{peak} to zero. In ImageNet, we use $\text{lr}_{\text{peak}} = 5 \times 10^{-4}$. In CC-15M, we use lower learning rate $\text{lr}_{\text{peak}} = 3 \times 10^{-4}$ and gradient clipping of 1.0 to avoid unstable training.

We select sampling hyper-parameters that achieve the best FID for the largest model for each benchmark. We use top-$k = 2048$ and softmax with temperature $t = 0.95$ for sampling in ImageNet if the decoding parameters are not specified. For rejection sampling in ImageNet, we use softmax with temperature $t = 0.95$ with acceptance rate 0.25. In CC-3M validation, we use softmax with temperature $t = 0.9$ for sampling. The performances with respect to sampling-related parameters (top-k, softmax temperature, acceptance ratio) are shown in Figure A6 and Figure A7.

Figure A1: Examples of reconstructed images with HQ-VAE with the learnable resampling layers. Reconstructed images at the top are processed by HQ-VAE trained without reconstruction losses for top code-only reconstruction images. Reconstructed images at the bottom are processed by HQ-VAE trained with the reconstruction losses for top code-only reconstruction images. Top code only reconstruction image at the top has a small amount of information, while the top code-only reconstruction image at the bottom shows a high-level structure with a distorted face.

A.3 Libraries and code repository

Our implementation is based on PyTorch 1.10 [2] with CUDA 11.3 and PyTorch Lightning 1.5.7 [3]. Major routines of our codes are based on the code repositories for VQ-GAN [4], minDALL-E [5] and RQ-Transformer [6].

B Ablation study

B.1 Details of resampling layers in HQ-VAE

We show the detailed analysis of resampling operation on HQ-VAE and its effect on HQ-Transformer in Table A2. We observe that both reconstruction FID (rFID) and top code only rFID are important for achieving good performance on stage 2. HQ-VAE with learnable resampling operations can be an example. We tested two additional loss functions, reconstruction and perceptual loss, on top code only reconstructed image for learnable resampling operations case. HQ-VAE with pixel reconstruction loss has lower rFID but high top code only rFID. It results in bad image generation performance compared to that of perceptual loss. HQ-VAE with pixel-shuffle achieve comparable performance on two rFIDs and it verifies the best generation performance supported by pixel-shuffle.

An example of top code collapsing for learnable resampling operations is shown at the top of Figure A1. Note that training with the learnable convolution layer is much slower than training with a non-learnable layer. HQ-VAE with the above learnable resampling layer takes about 300 minutes per epoch, while HQ-VAE with pixel-shuffle takes about 170 minutes per epoch with four NVIDIA Tesla A100 GPUs.

B.2 Input Embedding of HQ-Transformer

Ablation study (a) in Table 5 analyzes the effects on input embedding. We test two types of input embedding: adding and input embedding transformer (IET) in Eq (10). We introduce a simple adding

[2] PyTorch (`https://pytorch.org`)
[3] PyTorch Lightning (`https://www.pytorchlightning.ai`)
[4] VQ-GAN (`https://github.com/CompVis/taming-transformers`)
[5] minDALL-E (`https://github.com/kakaobrain/minDALL-E`)
[6] RQ-VAE and RQ-Transformer (`https://github.com/kakaobrain/rq-vae-transformer`)

Table A1: Hyperparamters for implementing HQ-Transformer.

Dataset	Model	# Params	N_{in}	N_{main}	N_{head}	L_{top}	L_{local}	d	d_e	n_e	τ
ImageNet	HQ-Transformer (S)	530 M	1	12	4	64	5	256	1536	24	1.0
	HQ-Transformer (M)	870 M	1	24	4	64	5	256	1536	24	1.0
	HQ-Transformer (L)	1437 M	1	42	6	64	5	256	1536	24	1.0
CC-15M	HQ-Transformer (S)	579 M	1	12	4	128	5	256	1536	24	1.0

Table A2: Ablation study on resampling operations on HQ-VAE and its effect on the generation performance of HQ-Transformer. We use sampling parameters top-k=1024 and τ=1.0 for sampling from transformer except the experiment with 50 epochs trained model.

Resampling Operation	epochs	rFID	rFID (top)	FID	Precision	Recall
Conv2 + Deconv2 (Pixel Perceptual Loss)	15	3.72	18.36	15.68	0.67	0.52
Conv2 + Deconv2 (Pixel Recon. Loss)	15	3.10	98.29	16.23	0.66	0.51
Avg. Pooling + NN interpolation	15	3.67	240.97	21.87	0.60	0.52
PixelShuffle + Unshuffle	15	3.14	83.45	13.59	0.67	0.52
PixelShuffle + Unshuffle	50	2.61	71.27	11.03	0.70	0.55

baseline for input embedding. Input embedding by adding is fusing embeddings for top and bottom codes and position embedding for top code. Channel-wise concatenation for embeddings of bottom codes is employed to reduce the parameter size and exploit the residual design where a single top code is aligned to r^2 bottom codes.

$$\mathbf{u}_i = \mathbf{e}_{\text{i}}^{\text{top,in}} + \text{Concatenation}\left(\mathbf{e}_{(i,1)}^{\text{bot,in}}, \cdots, \mathbf{e}_{(i,r^2)}^{\text{bot,in}}\right) + \text{PE}^{\text{top}}(i), \text{ for } i > 0 \tag{15}$$

However, we think that channel concatenation for bottom codes might cause information loss because adding embeddings of top code and concatenated embeddings of bottom codes requires a smaller channel dimension for each bottom code.

This is reason why we introduce IET, which has the equivalent channel dimension for both top and bottom codes. We use $L_{in} = 1, 2$ for the number of self-attention blocks in IET. We observe that a single layer for IET outperforms the other settings on FID and recall.

B.3 Prediction Head Transformer (PHT)

We propose parallel decoding for PHT and it assumes conditional independence among bottom codes given top code. As shown in Figure A2, typical causal attention observes the previous states only. So, it predicts code in a sequential manner. But, parallel decoding predicts a single top code and then predicts all bottom codes using contexts from the top colored in the red as shown in mixed attention of Figure A2. Ablation study (b) in Table A3 shows the effects on attention type for prediction head transformer (PHT).

Our parallel decoding has merit over causal attention in terms of speed. We measure the throughput of our models with diverse batch sizes as shown in Figure A5. Throughput for bidirectional decoding outperforms the others and throughput of causal decoding is the smallest one. Throughput of parallel decoding (mixed attention) is in the middle of two attention types. As the batch size becomes larger, the throughput gap between prediction head attention types becomes smaller.

Table A3: Ablation study on architecture of HQ-Transformer. We use the smallest model HQ-Transformer (S) to verify architectural choices.

Ablation	Input Embedding	Decoding Policy	Label Type	(top-k, t)	FID ↓	Precision ↑	Recall ↑
(a)	Adding	Parallel	One-hot label	$(2048, 0.9)$	11.03	0.70	0.55
	Transformer $N_{in} = 1$			$(2048, 0.95)$	10.01	0.69	0.55
	Transformer $N_{in} = 2$			$(2048, 1.0)$	11.86	0.70	0.51
(b)	Transformer $N_{in} = 1$	Causal	One-hot label	$(2048, 0.95)$	11.33	0.68	0.55
		Parallel			10.01	0.69	0.55
		Bidirectional			26.72	0.52	0.59
(c)	Transformer $N_{in} = 1$	Parallel	One-hot label	$(2048, 0.95)$	10.01	0.69	0.55
			Soft-label $\tau = 1$		9.36	0.69	0.55

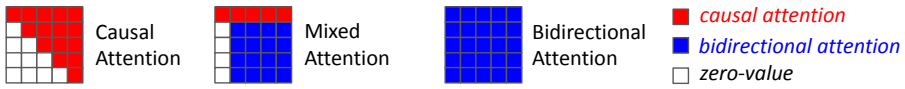


Figure A2: An example of causal, mixed, and bidirectional mask shapes for prediction head transformer with $r = 2$. We can implement a designated transformer by simply applying these mask patterns into masked self-attention blocks in the transformer.

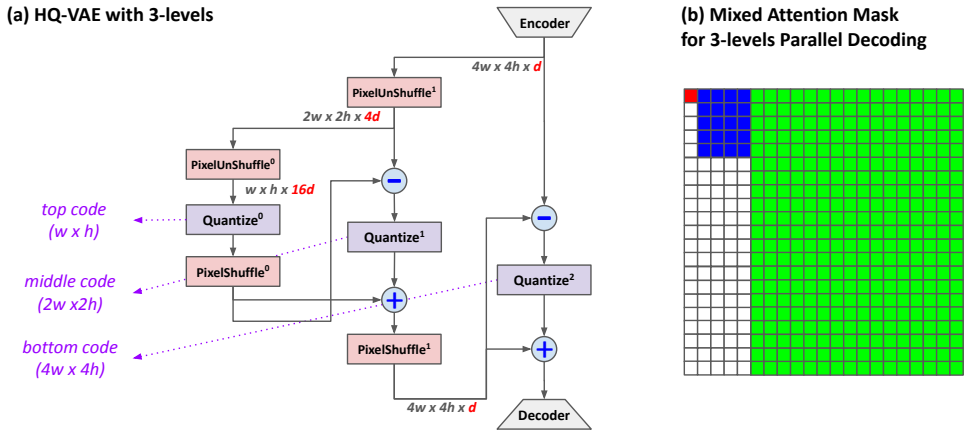


Figure A3: Multi-level extension for HQ-VAE (left) and mixed attention mask for three-level parallel decoding in HQ-Transformer (right). We use pixel-shuffle and -unshuffle for resampling operations as shown in the left. We recursively quantize hierarchical feature map to acquire three-level codes— top, middle and bottom. We also extend mixed attention mask to support parallel decoding for three-level codes as shown in the right.

B.4 Soft-labeling and softmax temperature in HQ-Transformer

Ablation study (c) in Table A3 examines the effect on soft-label and softmax temperature. As mentioned in the main paper, soft-label improves FID performance. We also observe that softmax temperature values below 1.0 improve FID performance if HQ-Transformer is trained with the soft-labeling scheme. The performances with respect to diverse parameters (top-k, softmax temperature, accentance ratio) are shown in Figure A6 and Figure A7.

B.5 Multi-level extension of HQ-VAE and HQ-Transformer

We extend two-level HQ-VAE into three-level and show the disentanglement of top and middle and bottom codes. The architecture of three-level HQ-VAE is shown in Figure A3 (left). Disentanglement in three-level HQ-VAE is visualized in Figure A4. It shows straightforward multi-level extension of HQ-VAE.

Locally hierarchical autoregressive model for three-level codes—top, middle and bottom codes—is also easily extended from two-level case. We introduce an additional conditional independence assumption among bottom codes given the associated top and middle codes for the multi-level extension as shown in the equation below. This multi-level extension of Eq 6 is simply given by applying the chain rule to Eq 5 with the middle code.

$$\begin{aligned}&\mathbb{P}_\theta(t_i, \mathbf{m}_i, \mathbf{b}_i | \mathbf{t}_{<i}, \mathbf{m}_{<i}, \mathbf{b}_{<i}) \\ =&\mathbb{P}_\theta(\mathbf{b}_i | t_i, \mathbf{m}_i, \mathbf{t}_{<i}, \mathbf{m}_{<i}, \mathbf{b}_{<i}) \cdot \mathbb{P}_\theta(\mathbf{m}_i | t_i, \mathbf{t}_{<i}, \mathbf{m}_{<i}, \mathbf{b}_{<i}) \cdot \mathbb{P}_\theta(t_i | \mathbf{t}_{<i}, \mathbf{m}_{<i}, \mathbf{b}_{<i})\end{aligned} \tag{16}$$

where, $\mathbf{b}_i = (b_{(i,1)}, \cdots b_{(i,4r^2)})$ and, $\mathbf{m}_i = (b_{(i,1)}, \cdots b_{(i,r^2)})$ HQ-Transformer with three-level code is easily implemented by plugging in a mixed attention mask for three-level parallel decoding in HQ-Transformer in Figure A3 (right).

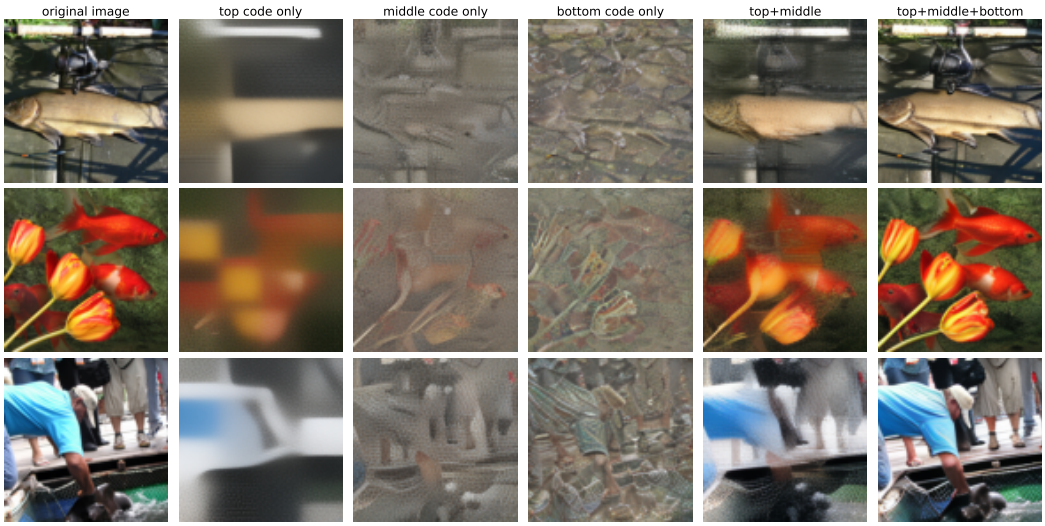


Figure A4: Visualization of reconstruction image of HQ-VAE with three-level codes with code resolution $(4 \times 4 + 8 \times 8 + 16 \times 16)$. The first column show (1) the original image and the other columns show reconstructed images synthesized from (2) only top code (3) only middle code (4) only bottom code (5) top and middle code (6) top, middle and bottom codes. Disentangled information in top, middle and bottom codes are clearly shown in the visualization.

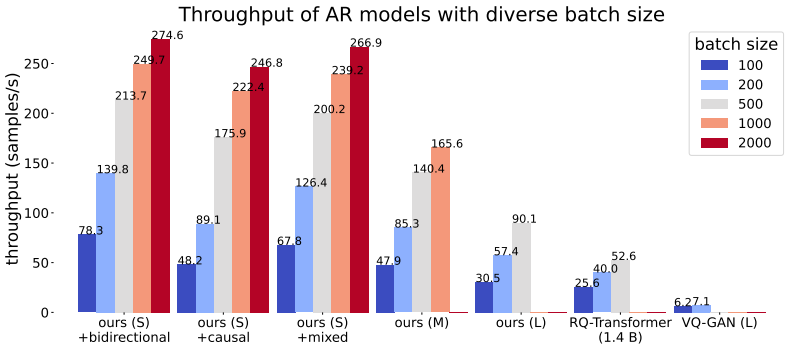


Figure A5: Comparison of throughput of AR models with diverse batch sizes. ours (M) and ours (L) are equipped with mixed attention in PHT.

Note that we use decoding parameter top-k = 2048 and temperature $(0.8^0, 0.8^1, 0.8^2)$ for three-level with code resolution $(4 \times 4 + 8 \times 8 + 16 \times 16)$ and temperature $(0.9^0, 0.9^1, 0.9^2)$ for three-level with code resolution $(8 \times 8 + 16 \times 16 + 32 \times 32)$. Setting low temperature for lower-level of codes improves the performance of three-level case. However, it does not improve the performance of two-level case.

C Additional image generation examples

We give additional samples for class-, text- and top code-conditional image generation. Figure A8 shows examples of class-conditional image generation Figure A9 shows examples of text-conditional image generation. The first row shows reconstructed images from the top code only. The second row shows reconstructed images from the bottom code only. The third row shows the final resulting images reconstructed from both the top and bottom code. Generated samples in each class and each prompt are acquired by fixing a random seed as 0.

Based on generated samples from class- and text-conditional image generation, we also conduct top code-conditional image generation shown in Figure A10 and Figure A11. Figure A10 shows top

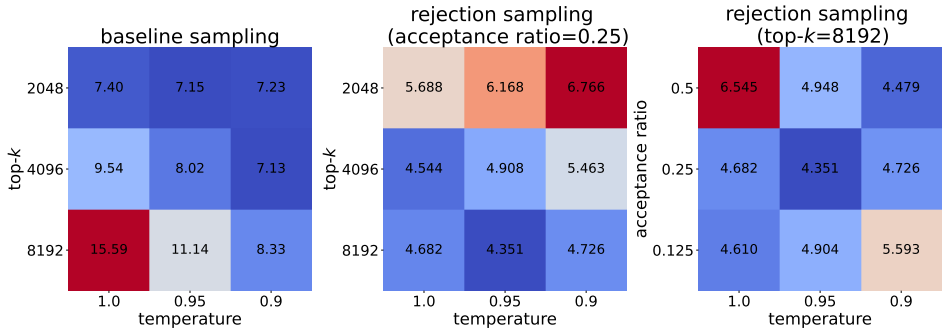


Figure A6: The performance of HQ-Transformer (L) on ImageNet training split measured by FID with diverse configurations with top-k, softmax temperature and acceptance ratio.

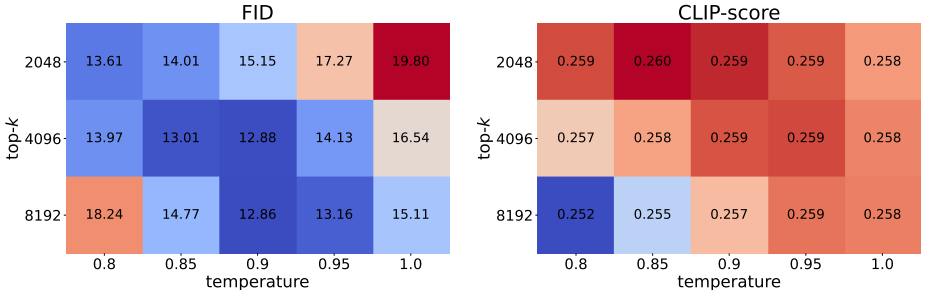


Figure A7: The performance of HQ-Transformer (S) on CC-3M validation split measured by FID and CLIP-score with diverse set of (top-k, softmax temperature).

code-conditional image generation by both class- and text-conditional generated images. Figure A11 shows 23 generated images from only the single given top code, which demonstrates the diversity in detail generation of our models.

D Limitations and Negative Societal Impacts

D.1 Limitations

Two-stage auto-regressive (AR) models have some artifacts in a small region with details. Figure A12 shows failure cases due to a scene with a small size of the face. Synthesized images only from top code in Figure A12 contain the solid color of the face rather than the rough structure of a human face, such as mouth, nose, or eyes. Solid color on the face without the structure might degrade details created with the independence assumption. To solve this problem, more information should be injected into the top code without distortion even in the small region. Note that other two-stage AR models [9, 18, 28] also experience the mentioned problems as reported in their supplementary materials. We think that introducing expressive visual codes is still a promising solution to this problem.

Multi-level extensions of our approaches have limitations over two-level baselines. First, multi-level HQ-VAE has lower effective top code usage, which results in poor performance compared to two-level counterparts. Second, the tuning on parallel decoding parameter is needed. It implies that we need better parallel decoding policy for multi-level extension.

D.2 Negative Societal Impacts

Unknown biases or unfiltered inappropriate samples in large-scale datasets might contribute to the generation of biased or harmful content and our method also shares the core of these problems. We believe that more research is needed on those problems for large-scale image generation.

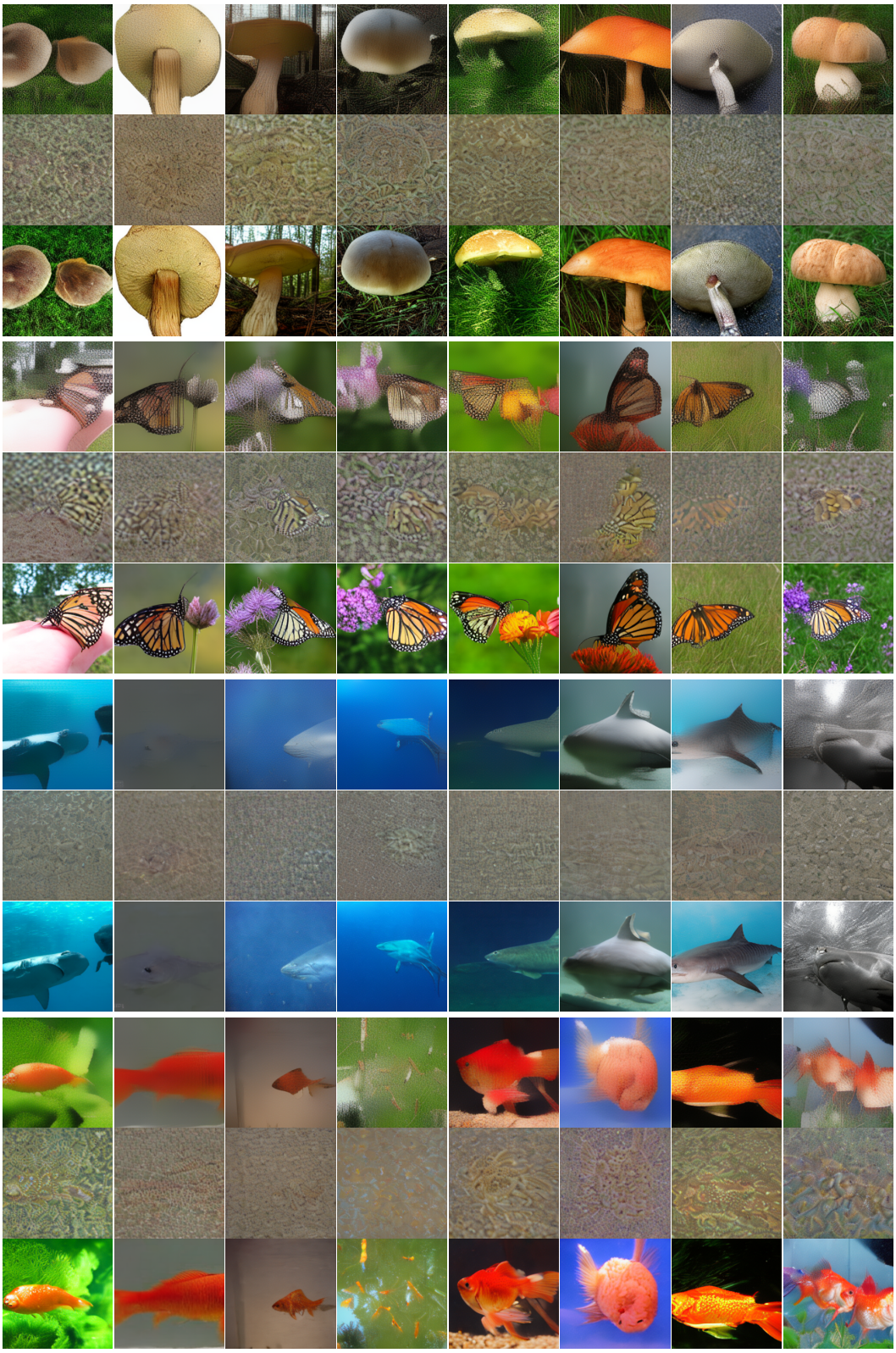


Figure A8: Examples of class-conditional image generation with ImageNet class *hen-of-the-woods* (996), *monarch butterfly* (323), *tiger shark* (3) and *goldfish* (1). The first row shows samples synthesized only from the top codes, the second row from the bottom codes only, and the third row shows our final generation samples.

Figure A9: Examples of text-conditional image generation from text prompt written in the title of each image. We visualize 8 samples out of 128 for each prompt using CLIP re-ranking. Measured CLIP score between an image with a text prompt is annotated at the top of each column. The first row shows samples synthesized only from the top codes, the second row from the bottom codes only, and the third row shows our final generation samples.

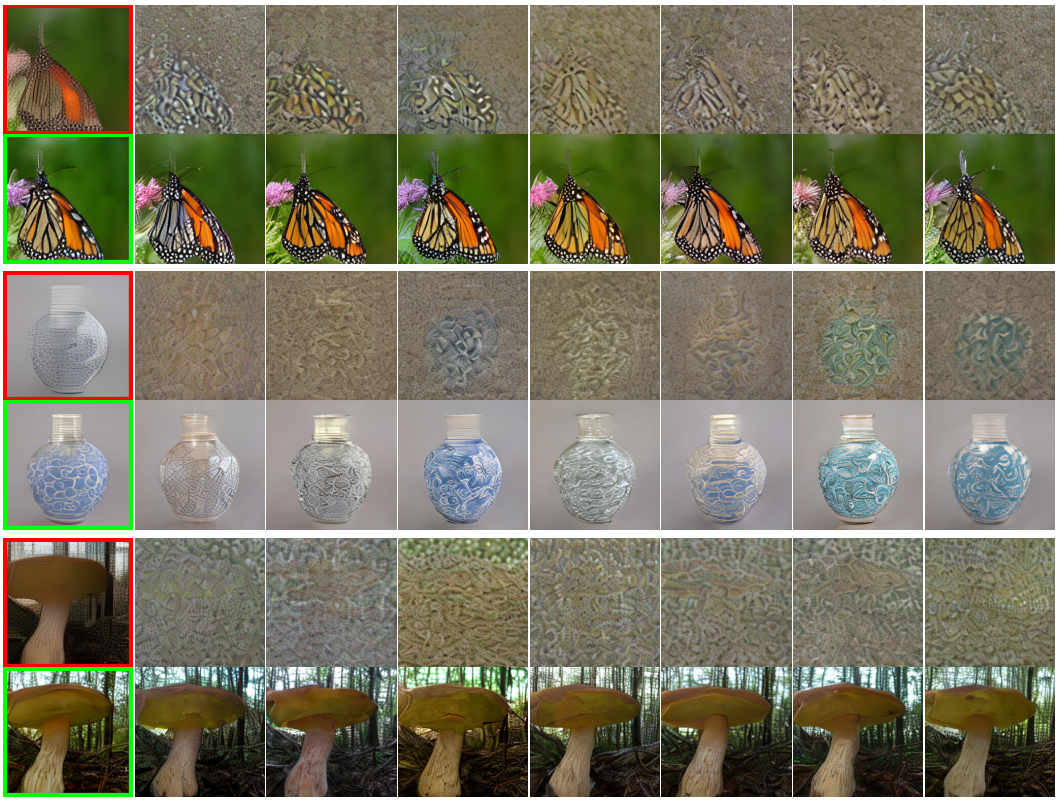


Figure A10: Examples of top code-conditional image generation with generated images by conditioning ImageNet class *monarch butterfly* (323), *vase* (883), *hen-of-the-woods* (996) shown in top-to-bottom. We extract the top codes from the green bordered images at below in each first column. The red bordered images visualize an image synthesized from the extracted top codes only. The samples on the right side of the red bordered one are synthesized only from the bottom codes. The samples on the right side of the green bordered are final results synthesized from the red-bordered top code and its upper side bottom codes.

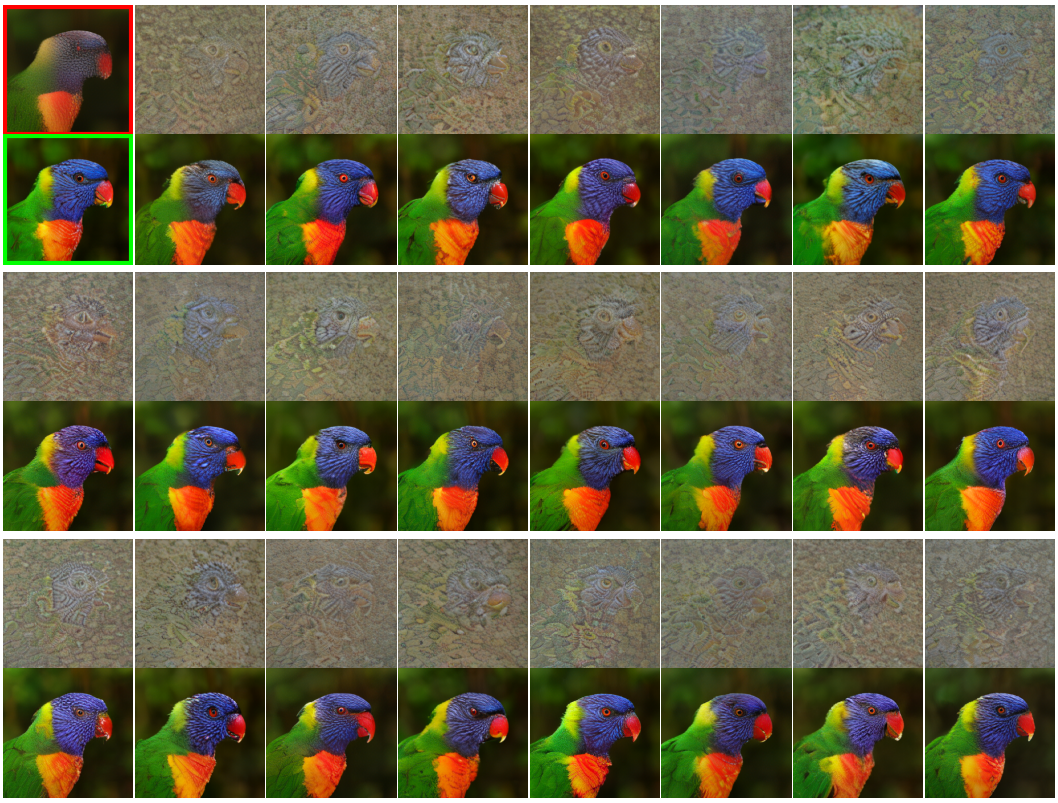


Figure A11: An example of 23 images generated from top code-conditional image generation with a single image of ImageNet class *lorikeet* (90). We extract the top codes from the green bordered image. The red bordered image visualizes an image reconstructed from the extracted top codes only. The samples on the odd-numbered row are synthesized only from the bottom codes. The samples on the even-numbered row are final results synthesized from the red-bordered top and bottom codes from its upper side. Every sample exhibits various details under the same structure given by the top code.

Figure A12: A failure case of class-conditional image generation from ImageNet class *doctor fish* or *tench* (0). The first row shows samples synthesized only from the top codes, the second row from the bottom codes only, and the third row shows our final generation samples. The structure of human face does not appear in the first row images reconstructed only from top code.