Locally Hierarchical Auto-Regressive Modeling for Image Generation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Auto-Regressive (AR) modeling on image generation provides a scalable solution to synthesize high-quality examples by directly optimizing the data likelihood, factorized into a product of conditionals in pre-defined generation order. However, it is computationally expensive to train AR models on the pixels of high-resolution images, and many practical approaches decompose a learning process into the following two stages: (a) representing a high-resolution image by a short discrete sequence, and (b) learning an AR model on this sequence, rather than on pixels. This paper points out that every code in the sequence has a receptive field of the same size and most existing works in this framework fail to exploit multi-scale information of images. To mitigate this limitation, we represent an image by a set of discrete sequences of multiple resolutions and generate images using an AR model based on variational autoencoder, which is referred to as Hierarchical-Quantized Variational AutoEncoder (HQ-VAE). We propose a Hierarchical-Quantized Transformers (HQ-Transformer) to model the multi-level discrete sequences efficiently and generate novel images of good quality. Experiments on class-conditional and text-conditional generation tasks verify that our approach outperforms previous AR models, in terms of fidelity of generation samples under a similar computation budget. We also demonstrate structure-condition image generation to show the usage of disentangled top and bottom codes. Extensive ablation studies explain rationales for our architectural design.

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].

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

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

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.

conditional needs to process many past observations. In the context of image generation, VQ-VAE [36] reduces the complexity of AR models by compressing a continuous RGB signal of high resolution into a discrete sequence of low resolution. In specific, VQ-VAE introduces a two-stage learning process, where stage 1 represents an image by a set of discrete codes, and stage 2 constructs AR priors [4, 35] on discrete sequences. 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. Many approaches to reducing the sequence length have been proposed to improve the stage 1 architecture [17, 38] or training objectives [9, 18]. These methods, however, adopt the codes with the same receptive field size. AR models in the second stage, therefore, need to process each code in the sequence equally important. 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 set of codes 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) to disentangle the information encoded in the top and bottom codes, without introducing additional losses and learnable components for disentanglement.
- We propose HQ-Transformer (Hierarchical-Quantized Transformer) that fully exploits the spatial relationship between the top and bottom codes. Our network is built upon an assumption that there exist several bottom codes conditionally independent given a top code.
- 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.

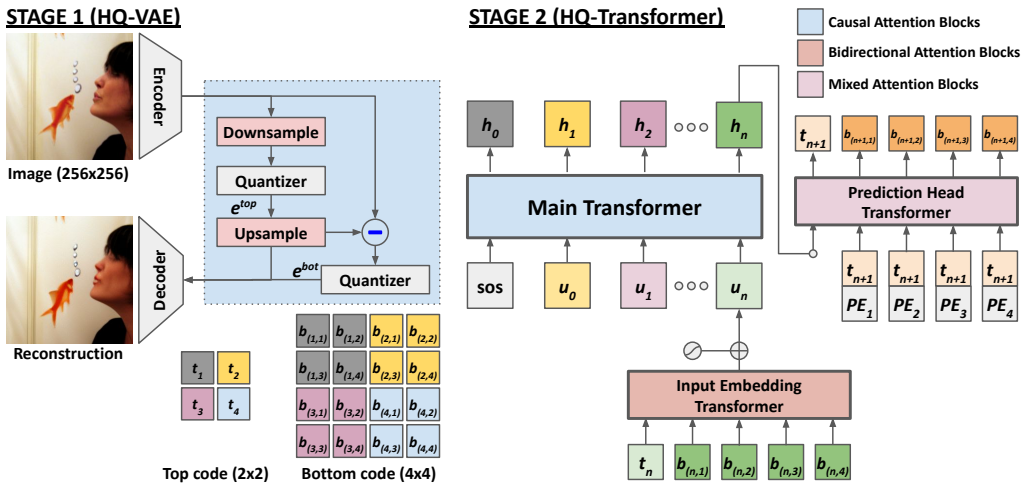


Figure 2: Conceptual illustration on the overall architecture: (a) the left one presents the components of HQ-VAE, where the residual and resampling operations enable to encode different information levels in the top and bottom codes, and (b) the right one shows the components of HQ-Transforms to train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

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. In order to do that, 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. RQ-VAE [18] further reduces the sequence length by converting an image into a stack of codes. 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. 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 the codes of multiple resolutions [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, but these require specialized modules to disentangle the information in the code levels. 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.

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.

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

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 $\mathbf{t}$ and $\mathbf{b}$ have the shapes of $l \times l$ and $rl \times rl$, respectively, with a scaling factor $r \geq 1$. The objective of this bottleneck process is to encode different levels of information using the top and bottom codes. Intuitively, it is desirable for the top codes to hold high-level information of an input image while low-level details are encoded by the bottom codes. To accomplish this objective, our hierarchical quantization adopts a two-stream network, with no additional learnable components.

In specific, given a scaling factor r and a feature map $\mathbf{z}$, the following process extracts the top code and its code embedding:

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

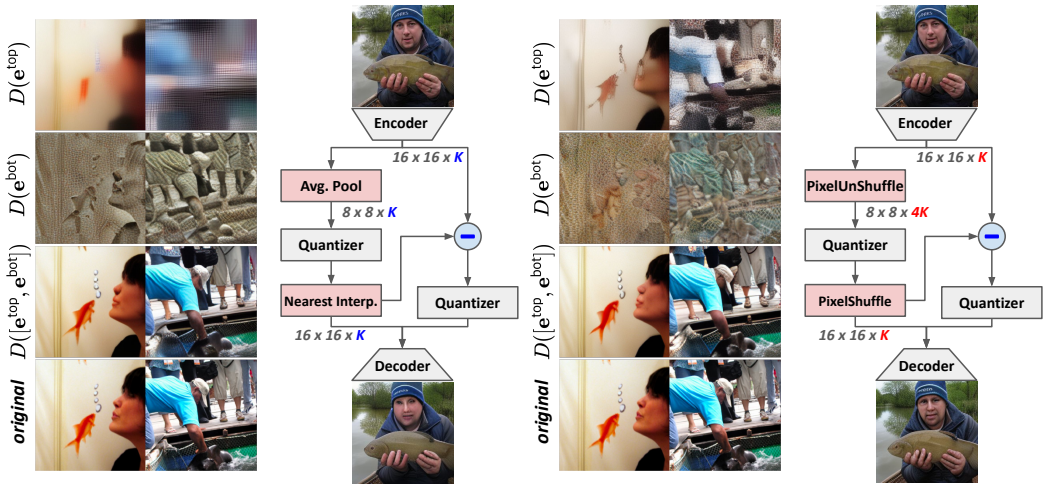
where $\mathcal{C}^{\text{top}}$ denotes the codebook of top codes and $\text{Downsample}(\cdot)$ is the average pooling function to reduce the spatial resolution of a feature map. Based on $\mathbf{t}$, the bottom code and its embedding are derived by

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

where $\mathcal{C}^{\text{bot}}$ is the codebook of bottom codes and $\text{Upsample}(\cdot)$ is implemented as an interpolation operation with the 'nearest' option. Figure 2(left) illustrates this hierarchical quantization process.

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 only contains high-frequency components in the feature map. Figure 3 compares the reconstructed images from each of the top and bottom code embeddings, demonstrating the disentangling performance by the proposed hierarchical vector quantization.

Despite the reasonable performance given by the resampling idea, 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.



(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.

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. We annotate all bottom codes aligned with a single top code t_i as $\mathbf{b}_i$. 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:

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

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

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

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

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

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:

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

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:

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

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:

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

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

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

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 $O(N_{\text{main}} L_{\text{top}}^2 r^4)$.

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.

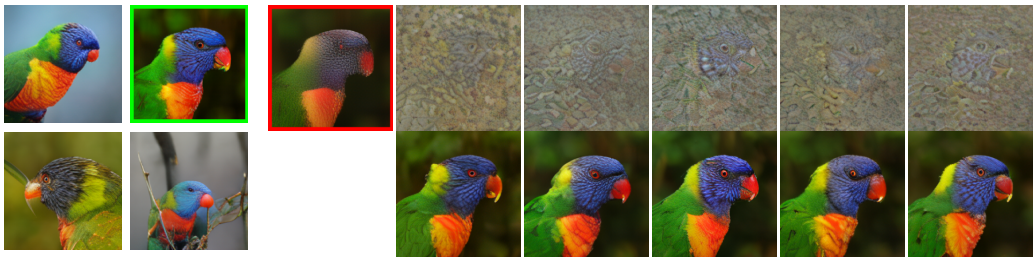
We setup HQ-VAE by $f = 8$ and $r = 2$ to produce the top and bottom codes of 8×8 and 16×16 resolution, respectively. We instantiate three types of HQ-Transformer: (a) $L_{\text{main}} = 12$, and $L_{\text{head}} = 4$ for the smallest model, (b) $L_{\text{main}} = 24$ and $L_{\text{head}} = 4$ for the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

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)	-	-	-
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	-
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.

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 650x 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),

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 ($f = 16, r = 2$) (ours)	$8 \times 8 + 16 \times 16$	2.61	8192 + 8192	8119 + 8192
HQ-VAE ($f = 8, r = 2$) (ours)	$16 \times 16 + 32 \times 32$	0.57	8192 + 8192	8192 + 8192

Table 4: Ablation study on HQ-VAE.

Resampling op.	Epochs	rFID ↓
Avg. Pool.	15	3.67
PixelShuffle	15	3.14
PixelShuffle	50	2.61

Table 5: Ablation study on HQ-Transformer (S)

Attention type	Label type	FID ↓
Causal	One-hot	11.17
Mixed		10.11
Mixed	Soft-label ($\tau = 1$)	9.36

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.

Ablation study on PHT in HQ-Transformer Table 5 compares the effects on the type of self-attention module and the target label type in Prediction Head Transformer. In Table 5, the causal attention means that the top and bottom codes are sequentially generated like in typical AR models. The mixed attention means our procedure; the top code is first generated and r^2 bottom codes are decoded in parallel. We observe that our parallel generation scheme achieves better FID than a fully-autoregressive policy, 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].

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.

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 Appendix

Optionally include extra information (complete proofs, additional experiments and plots) in the appendix. This section will often be part of the supplemental material.