Visual AutoRegressive Modeling (VAR)

Scalable Image Generation via Next-Scale Prediction

Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, Liwei Wang

Peking University & ByteDance | arXiv:2404.02905 | NeurIPS 2024

Table of Contents

Background: Why Autoregressive Modeling for Images?

The AR Recipe That Works

- Next-token prediction — predict each token given all prior tokens; core of GPT, LLaMA, PaLM
- Scaling laws — test loss follows a power-law with model size; bigger models always improve (Kaplan et al., 2020)
- Zero-shot generalization — models handle unseen tasks without fine-tuning
- Vision AR pipeline — VQVAE tokenizes images → flatten 2D grid to 1D → GPT-style next-token prediction

The issue on AR model with images

- VQGAN FID: 18.65 vs. DiT FID: 2.27 on ImageNet
- No demonstrated scaling laws for vision AR
- Raster-scan flattening breaks spatial structure

Figure 1: **Generated samples from Visual AutoRegressive (VAR) transformers trained on ImageNet**. We show 512×512 samples (top), 256×256 samples (middle), and zero-shot image editing results (bottom).

Key Question

Is the problem AR itself — or how to define the autoregressive order on images?

Why Vision AR Lags

On ImageNet 256×256, standard AR (VQGAN) achieves FID of **18.65**, while DiT-XL/2 reaches **2.27** — an 8× quality gap.

VQGAN (AR)

FID: 18.65

IS: 80.4

DiT-XL/2 (Diffusion)

FID: 2.27

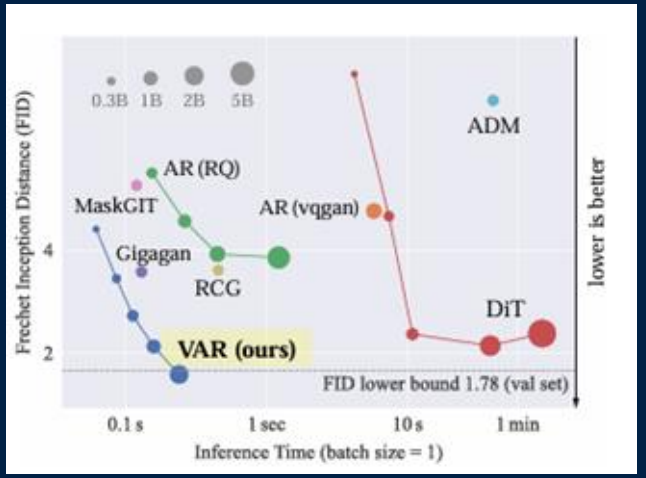
IS: 278.2

What's Missing from Vision AR?

- No demonstrated scaling laws
  — unlike LLMs where bigger models predictably improve
- No zero-shot generalization
  — models can't do in-painting, editing, or out-painting
- Extremely slow inference
  — $O(n^2)$ steps, $O(n^6)$ compute for n×n image tokens

FID vs. Inference Speed (ImageNet 256×256)

Model	FID ↓	IS ↑	#Step
VQGAN	18.65	80.4	256
ViTVQ-re	3.04	227.4	1024
MaskGIT	6.18	182.1	8
DiT-XL/2	2.27	278.2	250
L-DiT-7B	2.28	316.2	250



The Core Problem: Raster-Scan Flattening Breaks Images

From 2D Structure to 1D Sequence

Standard vision AR pipeline:

Four Issues with This Approach

1. **Mathematical Premise Violation**
   VQVAE encoder produces bidirectionally-correlated tokens (via self-attention). Flattening forces unidirectional dependency — a contradiction.
2. **Spatial Structure Destroyed**
   Token q(i,j) and its 4 neighbors are tightly correlated in 2D. In the flattened 1D sequence, they become distant — locality is lost.
3. **Limited Zero-Shot Generalization**
   Unidirectional ordering means the model cannot predict the top of an image given the bottom — blocking tasks like out-painting.
4. **Computational Inefficiency**
   Generating n×n tokens requires $O(n^2)$ autoregressive steps and $O(n^6)$ total computation — prohibitively slow.

Why Flattening Hurts: Spatial Locality

2D Token Grid (original):

1	2	3	4
5	★	7	8
9	10	11	12
13	14	15	16

★ and its 4 neighbours (2, 5, 7, 10) are adjacent

1D Flattened Sequence (raster-scan):

1	2	3	4	5	★	7	8	9	10	11	12	13	14	15	16

neighbour 10 is now 4 positions away from ★
→ Spatial correlation broken by flattening!

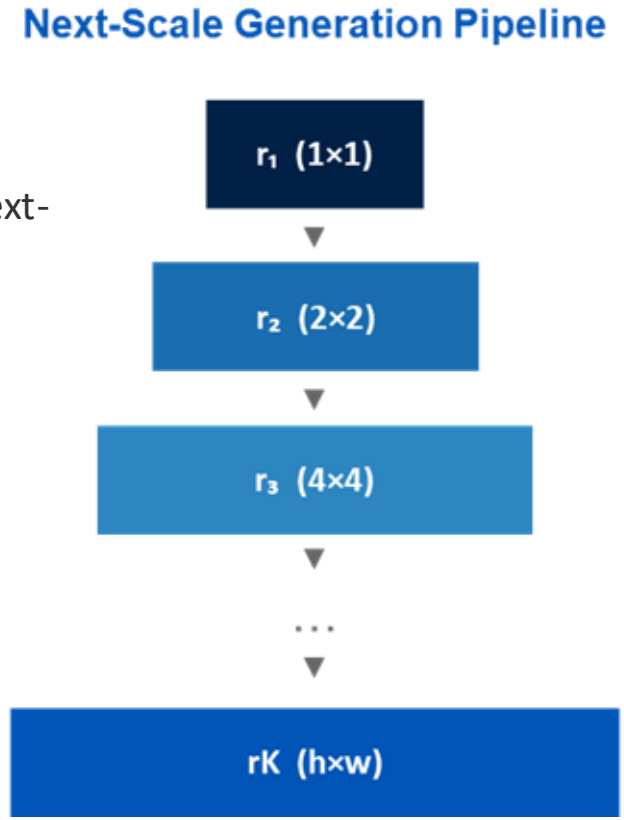
VAR Key Idea: Next-Scale Prediction

What is VAR?

Redefines autoregressive learning on images as coarse-to-fine **"next-scale prediction"**, replacing the standard raster-scan "next-token prediction".

How It Works (High Level)

- Encode image into multi-scale token maps (r_1, r_2, …, rK) at increasing resolutions
- Start from the coarsest scale (1×1 token map)
- At each step, predict the next higher-resolution token map conditioned on all previous maps
- All tokens within each scale generated in parallel



Decoder-only Transformer Inference Review

Input

Tokens

$$x_1, x_2, \ldots, x_t$$

$$e_1, e_2, \ldots, e_t \in \mathbb{R}^d$$

Word Embedding

Transformer Blocks

$$v_i = W_V e_i \in \mathbb{R}^{d_v}, \quad i = 1, \ldots, t$$

$$k_i = W_K e_i \in \mathbb{R}^{d_k}, \quad i = 1, \ldots, t$$

$$q_t = W_Q e_t \in \mathbb{R}^{d_k}$$

$$z = \text{softmax}\left(\begin{bmatrix} q_t^\top k_1 \\ q_t^\top k_2 \\ \vdots \\ q_t^\top k_t \end{bmatrix}\right) \in \mathbb{R}^t$$

$$V = \begin{bmatrix} v_1 & v_2 & \cdots & v_t \end{bmatrix} \in \mathbb{R}^{d_v \times t}$$

$$h_t = Vz \in \mathbb{R}^{d_v}$$

$$e_t \longrightarrow e_t + h_t$$

$$e_t \longrightarrow e_t + \text{MLP}(e_t)$$

Final Layer

$$\ell_t = W_{\text{vocab}} e_t + b$$

$$W_{\text{vocab}} \in \mathbb{R}^{V \times d}, \quad \ell_t \in \mathbb{R}^V$$

$$x_{t+1} \sim \text{Categorical}(\text{softmax}(\ell_t))$$

Image tokenization: VQVAE

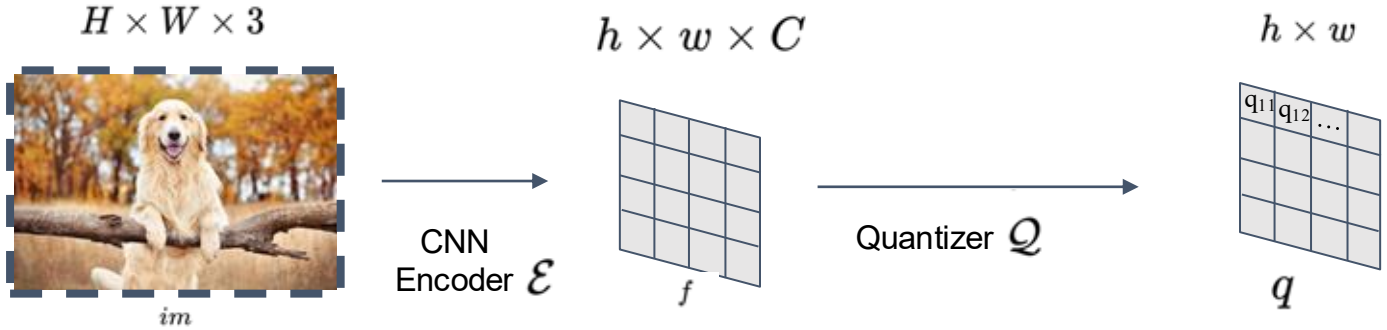


Image tokenization: VQVAE

$H = 16 \times h$

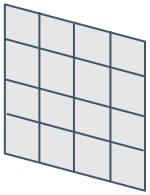
$W = 16 \times w$

$H \times W \times 3$

im

CNN Encoder $\mathcal{E}$

$h \times w \times C$



f

Quantizer $\mathcal{Q}$

$h \times w$

q

so that each C vector, and token,
represents a 16 by 16 image

Image tokenization: VQVAE

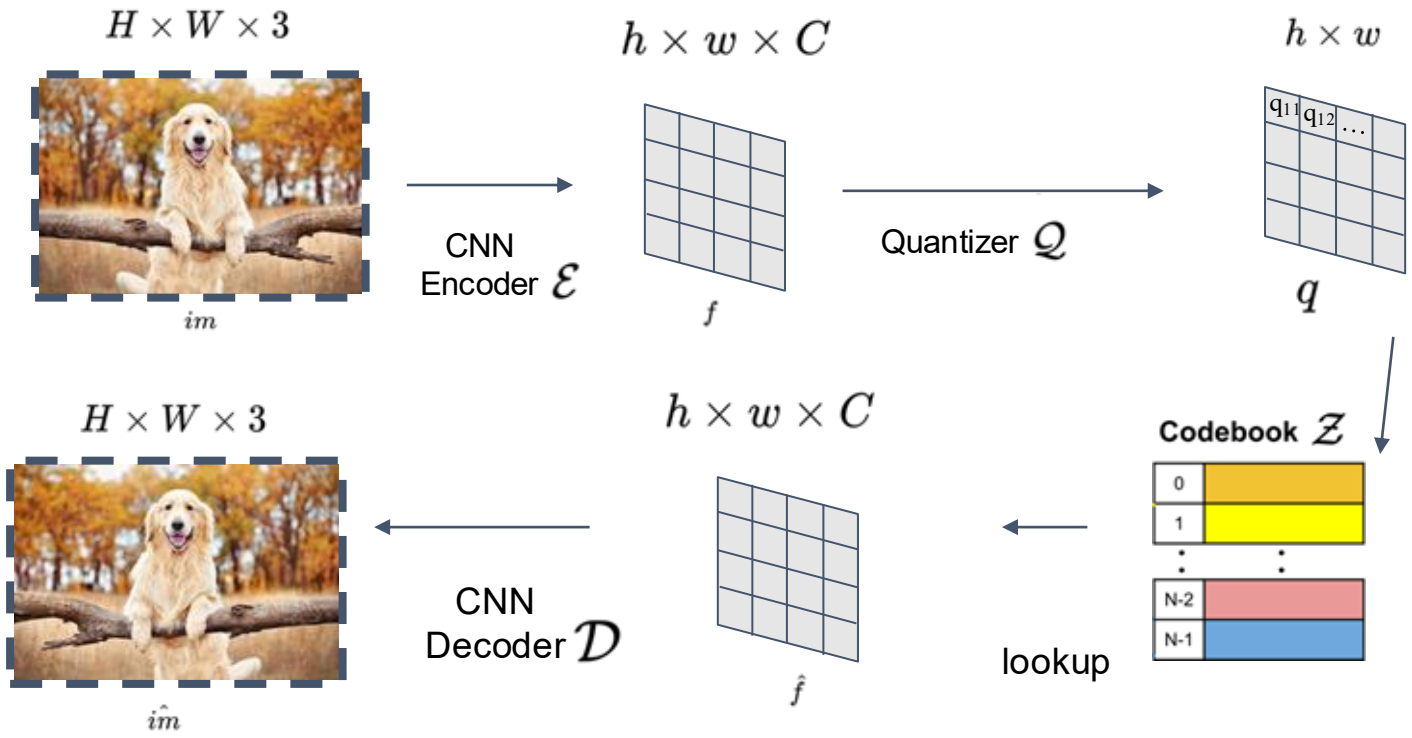
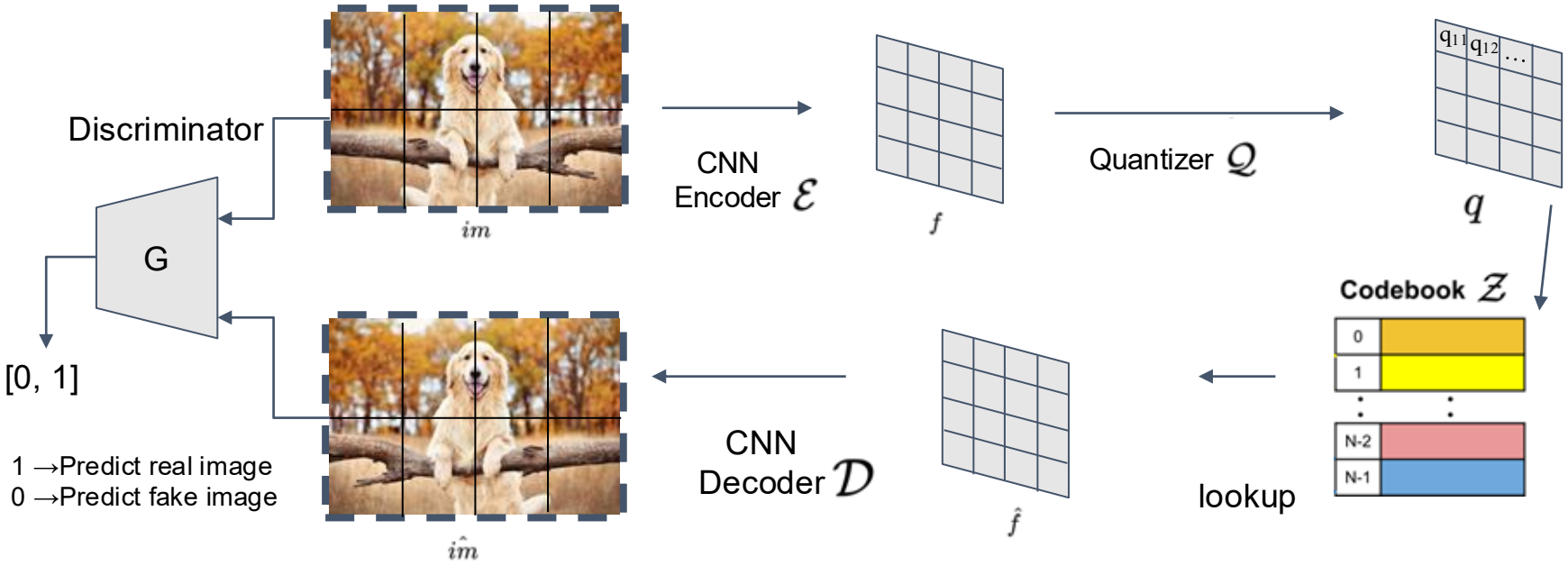
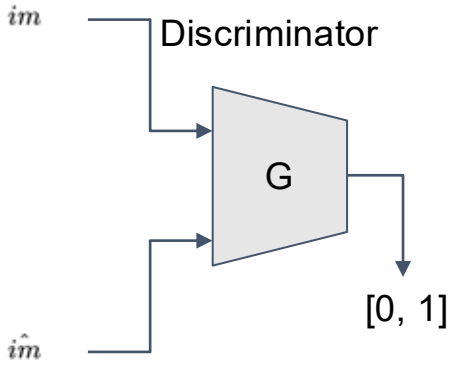


Image tokenization: GAN



$$\mathcal{L}_G = \left[\log G(im) + \log\left(1 - G(\hat{im})\right)\right] \qquad \min_{\mathcal{E},\mathcal{D},\mathcal{V}} \max_G \mathcal{L}_G$$

Image tokenization: GAN



1 →Predict image fake
0 →Predict image real

$$\min_{\mathcal{E},\mathcal{D},\mathcal{V}} \max_G \mathcal{L}_G$$

$$\mathcal{L}_G = \left[\log G(im) + \log\left(1 - G(i\hat{m})\right)\right]$$

Discriminator correctly identifies original and generated images

$$\mathcal{L}_G = [\log(1) + \log(1-0)] = 0$$

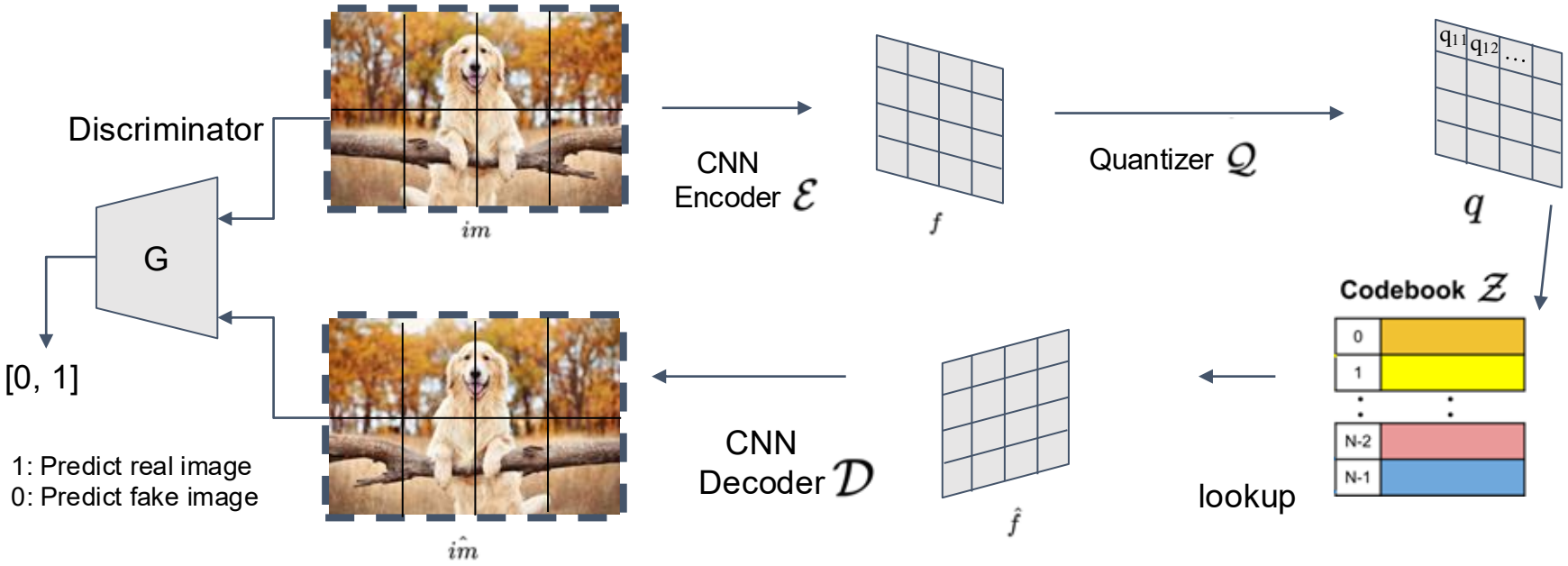
VQVAE tricks (min) discriminator

$$\mathcal{L}_G = [\log(0) + \log(1-1)] \to -\infty$$

(Discriminator spits out real values between 0 and 1, based on its confidence)

Therefore, this optimization problem learns a better image generator!

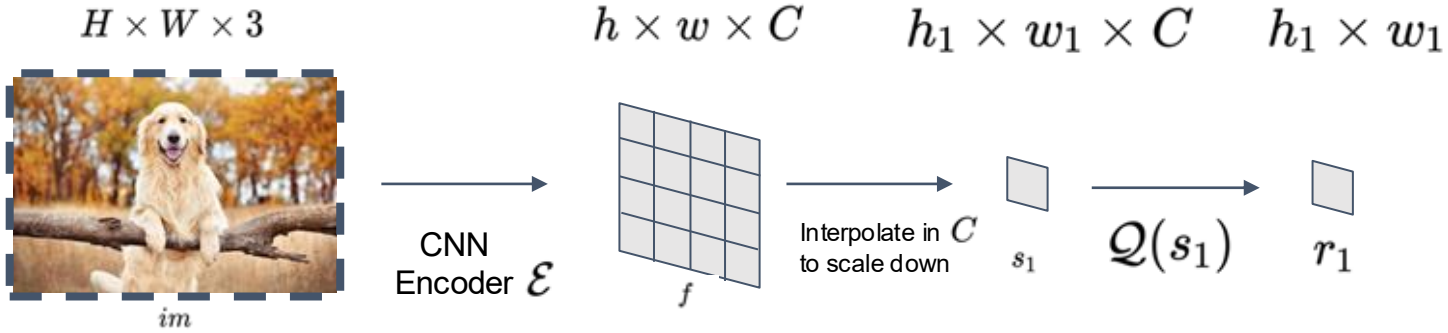
Image tokenization: VQVAE Training Loss



$$\mathcal{L} = \left\| im + i\hat{m} \right\|_2 + \left\| f - \hat{f} \right\|_2 + \lambda_P \mathcal{L}_P(i\hat{m}) + \lambda_G \mathcal{L}_G(i\hat{m})$$

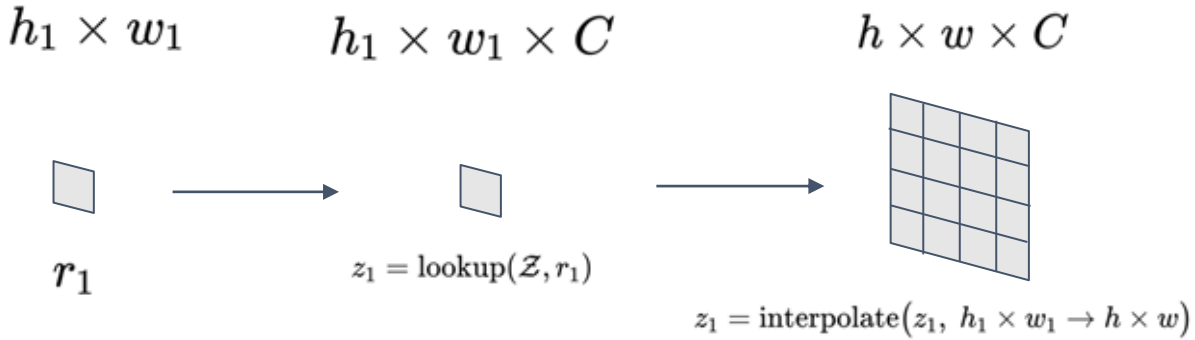
Multi Scale VQVAE

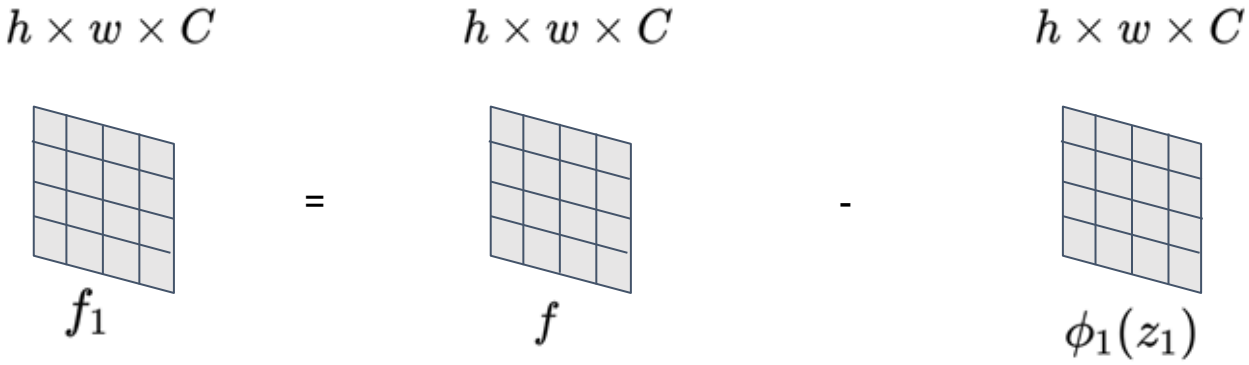
Scale 1 Tokenization



Multi Scale VQVAE

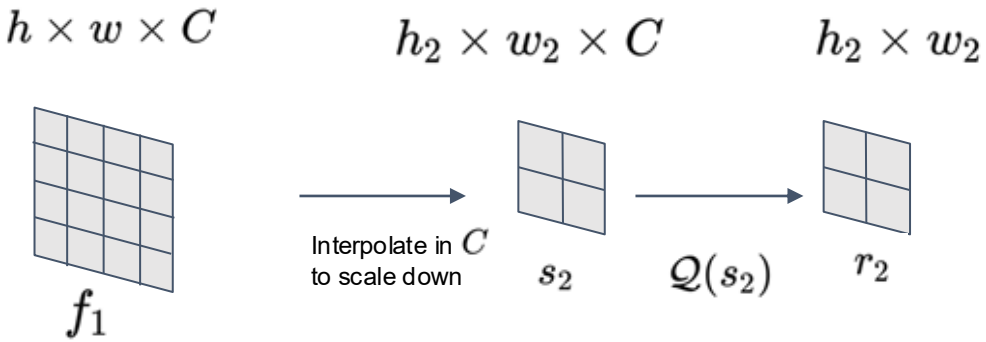
Upscale to update f





Multi Scale VQVAE

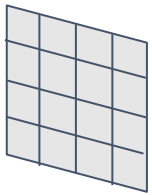
Scale 2 Tokenization



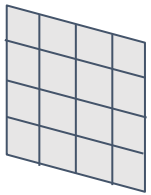
Multi Scale VQVAE

Scale 2 Tokenization

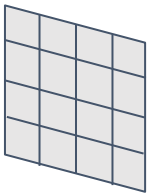
$h \times w \times C$ $h \times w \times C$ $h \times w \times C$



f_2 =



f_1 -



$\phi_2(z_2)$

$$z_2 = \text{interpolate}\big(\text{lookup}(\mathcal{Z}, r_2),\ h_1 \times w_1 \to h \times w\big)$$

Multi Scale VQVAE

Tokenization Algorithm

Algorithm 1: Multi-scale VQVAE Encoding

Inputs: raw image im;
Hyperparameters: steps K, resolutions $(h_k, w_k)_{k=1}^{K}$;
$f = \mathcal{E}(im)$, $R = []$;
for $k = 1, \cdots, K$ **do**
  $r_k = \mathcal{Q}(\text{interpolate}(f, h_k, w_k))$;
  $R = \text{queue\_push}(R, r_k)$;
  $z_k = \text{lookup}(Z, r_k)$;
  $z_k = \text{interpolate}(z_k, h_K, w_K)$;
  $f = f - \phi_k(z_k)$;
Return: multi-scale tokens R;

- Interpolate the latent representation f to the desired scale
- Quantize it element-wise to get the token map at scale k
- Map the tokenized scale back to the image latent space and subtract it from f, for interpolation at the next scale

Multi Scale VQVAE

Reconstruction Algorithm

Algorithm 2: Multi-scale VQVAE Reconstruction

Inputs: multi-scale token maps R;
Hyperparameters: steps K, resolutions $(h_k, w_k)_{k=1}^{K}$;
$\hat{f} = 0$;
for $k = 1, \cdots, K$ **do**
    $r_k = \text{queue\_pop}(R)$;
    $z_k = \text{lookup}(Z, r_k)$;
    $z_k = \text{interpolate}(z_k, h_K, w_K)$;
    $\hat{f} = \hat{f} + \phi_k(z_k)$;
$\hat{im} = \mathcal{D}(\hat{f})$;
Return: reconstructed image $\hat{im}$;

- Reverse the tokenization operation: construct $\hat{f}$ from the multi-scale token maps
- Initialize $\hat{f}$ to zero. Each iteration, scale up the token data in the latent space using interpolation and apply phi_k cnns, to address information loss on upscaling, and ad $\hat{f}$ to
- Apply VQVAE decoder to get resulting image

VAR

Transformer Input

Recall…

$x_1, x_2, \ldots, x_t$

↓

$e_1, e_2, \ldots, e_t \in \mathbb{R}^d$

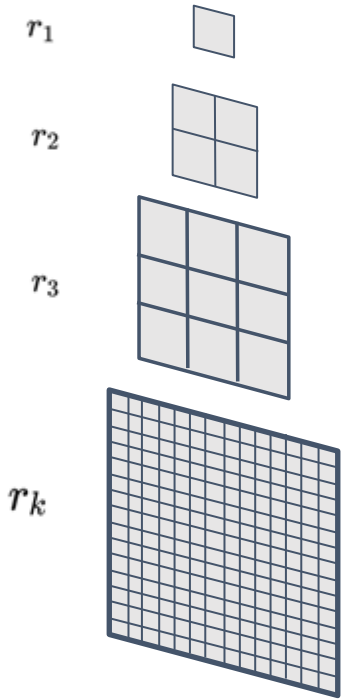
↓

Transformer

↓

$x_2, x_3, \ldots, x_{t+1}$

Token Maps



→

Token Sequence

q^1

$q_1^2, q_2^2, q_3^2, q_4^2$

$q_1^3, q_2^3, q_3^3, q_4^3, q_5^3, q_6^3, q_7^3, q_8^3, q_9^3$

$q_1^k, q_2^k, \ldots, q_{k^2}^k$

VAR

Transformer Input Upscaling

$r_1, r_2, \ldots, r_k$

↓

Transformer

↓

$r_2, r_3, \ldots, r_{k+1}$

$q^1, q_1^2, q_2^2, q_3^2, q_4^2$

↓

Transformer?

↓

$q_1^2, q_2^2, q_3^2, q_4^2, q_1^3, q_2^3, q_3^3, q_4^3, q_5^3, q_6^3, q_7^3, q_8^3, q_9^3$

VAR

Transformer Input Embedding

Given $r_1, r_2, \ldots, r_k$

We start by decoding, in the image latent space, the tokens we have so far to make an image in the latent space comprised of all the (lower) scale information we have up to this point:

$$\hat{f}_k = \sum_{j=1}^{k} \phi_j \big(\text{interpolate}\big(\text{lookup}(\mathcal{Z}, r_j),\ h_j \times w_j \to h \times w\big)\big)$$

Then, in the image latent space, we interpolate to the number of tokens in the next token map r_{k+1}

$$a_1^k, \ldots, a_{(k+1)^2}^k = \text{interpolate}\big(\hat{f}_k,\ h \times w \to h_{k+1} \times w_{k+1}\big) \quad a_i^k \in C$$

Instead of quantizing each of these elements, we learn an linear map from this space to the transformer's embedding space directly:

$$e_i^k = W_{\text{embed}}\, a_i^k$$

VAR

Transformer Input Upscaling

$r_1, r_2, \ldots, r_k$

↓

Transformer

↓

$r_2, r_3, \ldots, r_{k+1}$

$q^1, q_1^2, q_2^2, q_3^2, q_4^2$

↓ Decode Tokens in Latent Space

$\hat{f}_1, \hat{f}_2$

↓ Interpolate to token map dimensions
Map to transformer word embedding

$e_1^1, e_2^1, e_3^1, e_4^1, e_1^2, e_2^2, e_3^2, e_4^2, e_5^2, e_6^2, e_7^2, e_8^2, e_9^2$

↓

Transformer!

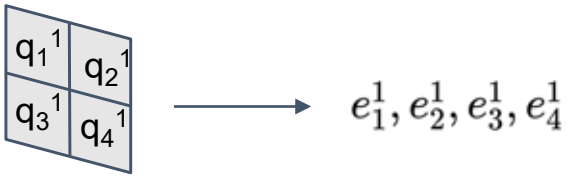
↓

$q_1^2, q_2^2, q_3^2, q_4^2, q_1^3, q_2^3, q_3^3, q_4^3, q_5^3, q_6^3, q_7^3, q_8^3, q_9^3$

VAR

Positional Encoding

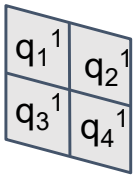
- We have converted a grid at a specific scale into a sequence… how do we recover locality?



VAR

Positional Encoding

- We have converted a grid at a specific scale into a sequence… how do we recover locality?
- We add positional embeddings!
- This is a vector in the embedding space, each learned separately



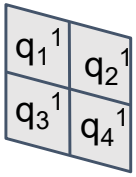
$$e_1^1, e_2^1, e_3^1, e_4^1$$

$$x_i = e_i + p_i, \quad e_i, p_i \in \mathbb{R}^d, \; d = 1024$$

VAR

Positional Encoding

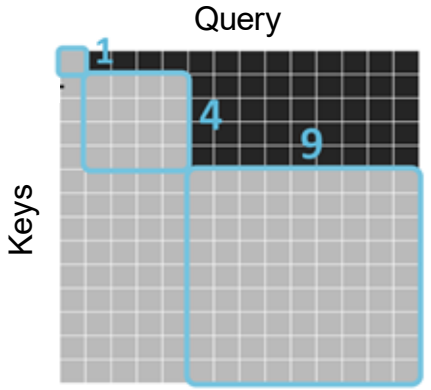
- We have converted a grid at a specific scale into a sequence… how do we recover locality?
- We add positional embeddings!
- This is a vector in the embedding space, each learned separately
- We also let each token in the block attend to every other token in the block



$$e_1^1, e_2^1, e_3^1, e_4^1$$

$$x_i = e_i + p_i, \quad e_i, p_i \in \mathbb{R}^d, \; d = 1024$$

Block-wise causal mask



VAR

Start of sequence, Classifier Free Guidance

- ImageNet dataset: 1.28 million images, 1K classes
- For the first sequence, we upscale to number of tokens in r_1 (typically 1) and learn an embedding for each class
- We also use classifier free guidance: Add an “unconditioned” class with its own embedding that is trained on every image class
  - Output logits: $\ell_{\text{cfg}} = \ell_{\text{uncond}} + 2(\ell_{\text{cond}} - \ell_{\text{uncond}}) = 2\ell_{\text{cond}} - \ell_{\text{uncond}}$

VAR

Other engineering hacks

- Adaptive Linear Norm: Learn gate, shift, and scale vectors for attention and mlp layers

$$g_{\text{attn}}(c) \cdot \text{Attn}\big(\gamma_{\text{attn}}(c) \cdot \text{LN}(H) + \beta_{\text{attn}}(c)\big)$$
$$g_{\text{mlp}}(c) \cdot \text{MLP}\big(\gamma_{\text{mlp}}(c) \cdot \text{LN}(H) + \beta_{\text{mlp}}(c)\big)$$

- This is also conditioned on “c”, the class label, through a small mlp
- Attention Normalization: normalize query and key vectors before computing attention
- Top-k: Use only the top-k logits in choosing the next token

Implementation

VAR tokenizer

- Vanilla **VQ-VAE tokenizer** with **multi-scale quantization**
- Only **K extra convolutions so** very small overhead: **0.03M parameters**
- **Shared codebook** across scales, **V = 4096**
- Tokenizer trained on **OpenImages**, **16× downsampling**

VAR transformer

- Standard **decoder-only transformer similar to GPT-2/VQGAN** with **Adaptive Layer Normalization (AdaLN)**.
- Class embedding used as **start token** and **AdaLN condition**
- **Normalized Q/K vectors** for stable attention
- No **RoPE**, **SwiGLU**, or **RMSNorm**

Implementation

VAR tokenizer

- Vanilla **VQ-VAE tokenizer** with **multi-scale quantization**
- Only **K extra convolutions so** very small overhead: **0.03M parameters**
- **Shared codebook** across scales, **V = 4096**
- Tokenizer trained on **OpenImages**, **16× downsampling**

VAR transformer

- Training is also fairly standard: AdamW, large batch sizes, and 200–350 epochs depending on model size.
    - base LR 10^{-4}
    - AdamW with $\beta_1 = 0.9, \beta_2 = 0.95$
    - weight decay 0.05
    - batch size 768–1024

Does VAR generate better images than prior methods?

How do we evaluate them?

- **FID (Fréchet Inception Distance):** Measures how close the generated image distribution is to real images; **lower is better**.
- **IS (Inception Score):** Measures both image quality and diversity based on classifier confidence; **higher is better**.
- **Precision:** Measures how realistic and high-quality the generated images are; **higher is better**.
- **Recall:** Measures how well the generated images cover the diversity of the real data distribution; **higher is better**.
- **#Step:** The number of steps required to produce one image; **lower means faster generation**.
- **Inference time:** The wall-clock time for image generation; **lower is better**.

Does VAR generate better images than prior methods?

Comparison on ImageNet 256×256

- The compared families include
  - **GANs**
  - **Diffusion models**
  - **Masked models**
  - **Traditional autoregressive (AR) models**
  - Proposed **VAR** models

Type	Model	FID↓	IS↑	Pre↑	Rec↑	#Para	#Step	Time
GAN	BigGAN [13]	6.95	224.5	**0.89**	0.38	112M	1	–
GAN	GigaGAN [42]	3.45	225.5	0.84	**0.61**	569M	1	–
GAN	StyleGan-XL [74]	2.30	265.1	0.78	0.53	166M	1	0.3 [74]
Diff.	ADM [26]	10.94	101.0	0.69	0.63	554M	250	168 [74]
Diff.	CDM [36]	4.88	158.7	–	–	–	8100	–
Diff.	LDM-4-G [70]	3.60	247.7	–	–	400M	250	–
Diff.	DiT-L/2 [63]	5.02	167.2	0.75	0.57	458M	250	31
Diff.	DiT-XL/2 [63]	2.27	278.2	0.83	0.57	675M	250	45
Diff.	L-DiT-3B [3]	2.10	304.4	0.82	0.60	3.0B	250	>45
Diff.	L-DiT-7B [3]	2.28	316.2	0.83	0.58	7.0B	250	>45
Mask.	MaskGIT [17]	6.18	182.1	0.80	0.51	227M	8	0.5 [17]
Mask.	RCG (cond.) [51]	3.49	215.5	–	–	502M	20	1.9 [51]
AR	VQVAE-2† [68]	31.11	~45	0.36	0.57	13.5B	5120	–
AR	VQGAN† [30]	18.65	80.4	0.78	0.26	227M	256	19 [17]
AR	VQGAN [30]	15.78	74.3	–	–	1.4B	256	24
AR	VQGAN-re [30]	5.20	280.3	–	–	1.4B	256	24
AR	ViTVQ [92]	4.17	175.1	–	–	1.7B	1024	>24
AR	ViTVQ-re [92]	3.04	227.4	–	–	1.7B	1024	>24
AR	RQTran. [50]	7.55	134.0	–	–	3.8B	68	21
AR	RQTran.-re [50]	3.80	323.7	–	–	3.8B	68	21
VAR	VAR-*d*16	3.30	274.4	0.84	0.51	310M	10	0.4
VAR	VAR-*d*20	2.57	302.6	0.83	0.56	600M	10	0.5
VAR	VAR-*d*24	2.09	312.9	0.82	0.59	1.0B	10	0.6
VAR	VAR-*d*30	1.92	323.1	0.82	0.59	2.0B	10	1
VAR	VAR-*d*30-re	**1.73**	**350.2**	0.82	0.60	2.0B	10	1
	(validation data)	1.78	236.9	0.75	0.67			

Does VAR generate better images than prior methods?

Comparison on ImageNet 256×256

- **VAR-d30-re** achieves the **best overall result** with **FID = 1.73** and **IS = 350.2**, outperforming all other compared methods.
- Even without rejection sampling, **VAR-d30** achieves **FID = 1.92** and **IS = 323.1**, which is still better than most diffusion and autoregressive baselines.

Type	Model	FID↓	IS↑	Pre↑	Rec↑	#Para	#Step	Time
GAN	BigGAN [13]	6.95	224.5	**0.89**	0.38	112M	1	–
GAN	GigaGAN [42]	3.45	225.5	0.84	**0.61**	569M	1	–
GAN	StyleGan-XL [74]	2.30	265.1	0.78	0.53	166M	1	0.3 [74]
Diff.	ADM [26]	10.94	101.0	0.69	0.63	554M	250	168 [74]
Diff.	CDM [36]	4.88	158.7	–	–	–	8100	–
Diff.	LDM-4-G [70]	3.60	247.7	–	–	400M	250	–
Diff.	DiT-L/2 [63]	5.02	167.2	0.75	0.57	458M	250	31
Diff.	DiT-XL/2 [63]	2.27	278.2	0.83	0.57	675M	250	45
Diff.	L-DiT-3B [3]	2.10	304.4	0.82	0.60	3.0B	250	>45
Diff.	L-DiT-7B [3]	2.28	316.2	0.83	0.58	7.0B	250	>45
Mask.	MaskGIT [17]	6.18	182.1	0.80	0.51	227M	8	0.5 [17]
Mask.	RCG (cond.) [51]	3.49	215.5	–	–	502M	20	1.9 [51]
AR	VQVAE-2† [68]	31.11	~45	0.36	0.57	13.5B	5120	–
AR	VQGAN† [30]	18.65	80.4	0.78	0.26	227M	256	19 [17]
AR	VQGAN [30]	15.78	74.3	–	–	1.4B	256	24
AR	VQGAN-re [30]	5.20	280.3	–	–	1.4B	256	24
AR	ViTVQ [92]	4.17	175.1	–	–	1.7B	1024	>24
AR	ViTVQ-re [92]	3.04	227.4	–	–	1.7B	1024	>24
AR	RQTran. [50]	7.55	134.0	–	–	3.8B	68	21
AR	RQTran.-re [50]	3.80	323.7	–	–	3.8B	68	21
VAR	VAR-*d16*	3.30	274.4	0.84	0.51	310M	10	0.4
VAR	VAR-*d20*	2.57	302.6	0.83	0.56	600M	10	0.5
VAR	VAR-*d24*	2.09	312.9	0.82	0.59	1.0B	10	0.6
VAR	VAR-*d30*	1.92	323.1	0.82	0.59	2.0B	10	1
VAR	VAR-*d30*-re	**1.73**	**350.2**	0.82	0.60	2.0B	10	1
	(validation data)	1.78	236.9	0.75	0.67			

Does VAR generate better images than prior methods?

Comparison on ImageNet 256×256

- Among GANs, **StyleGAN-XL** performs strongly with **FID = 2.30** and **IS = 265.1**, but it is still worse than the top VAR models in both metrics.
- Among diffusion models, **DiT-XL/2** achieves **FID = 2.27** and **IS = 278.2**, while larger models such as **L-DiT-3B** and **L-DiT-7B** also remain behind **VAR-d30-re** in FID.
- The masked model **MaskGIT** is faster than diffusion and traditional AR models, but its **FID = 6.18** is much worse than VAR.

Type	Model	FID↓	IS↑	Pre↑	Rec↑	#Para	#Step	Time
GAN	BigGAN [13]	6.95	224.5	**0.89**	0.38	112M	1	–
GAN	GigaGAN [42]	3.45	225.5	0.84	**0.61**	569M	1	–
GAN	StyleGan-XL [74]	2.30	265.1	0.78	0.53	166M	1	0.3 [74]
Diff.	ADM [26]	10.94	101.0	0.69	0.63	554M	250	168 [74]
Diff.	CDM [36]	4.88	158.7	–	–	–	8100	–
Diff.	LDM-4-G [70]	3.60	247.7	–	–	400M	250	–
Diff.	DiT-L/2 [63]	5.02	167.2	0.75	0.57	458M	250	31
Diff.	DiT-XL/2 [63]	2.27	278.2	0.83	0.57	675M	250	45
Diff.	L-DiT-3B [3]	2.10	304.4	0.82	0.60	3.0B	250	>45
Diff.	L-DiT-7B [3]	2.28	316.2	0.83	0.58	7.0B	250	>45
Mask.	MaskGIT [17]	6.18	182.1	0.80	0.51	227M	8	0.5 [17]
Mask.	RCG (cond.) [51]	3.49	215.5	–	–	502M	20	1.9 [51]
AR	VQVAE-2† [68]	31.11	~45	0.36	0.57	13.5B	5120	–
AR	VQGAN† [30]	18.65	80.4	0.78	0.26	227M	256	19 [17]
AR	VQGAN [30]	15.78	74.3	–	–	1.4B	256	24
AR	VQGAN-re [30]	5.20	280.3	–	–	1.4B	256	24
AR	ViTVQ [92]	4.17	175.1	–	–	1.7B	1024	>24
AR	ViTVQ-re [92]	3.04	227.4	–	–	1.7B	1024	>24
AR	RQTran. [50]	7.55	134.0	–	–	3.8B	68	21
AR	RQTran.-re [50]	3.80	323.7	–	–	3.8B	68	21
VAR	VAR-*d*16	3.30	274.4	0.84	0.51	310M	10	0.4
VAR	VAR-*d*20	2.57	302.6	0.83	0.56	600M	10	0.5
VAR	VAR-*d*24	2.09	312.9	0.82	0.59	1.0B	10	0.6
VAR	VAR-*d*30	1.92	323.1	0.82	0.59	2.0B	10	1
VAR	VAR-*d*30-re	**1.73**	**350.2**	0.82	0.60	2.0B	10	1
	(validation data)	1.78	236.9	0.75	0.67			

Does VAR generate better images than prior methods?

Comparison on ImageNet 256×256

- Traditional AR models such as **VQGAN**, **ViTVQ**, and **RQ-Transformer** generally require many more generation steps and have worse FID than VAR.
- All reported VAR models require only **10 steps**, while diffusion models typically require **250 steps**, and some AR models require up to **1024** steps.
- In terms of wall-clock inference time, **VAR-d16** takes only **0.4×**, **VAR-d20** takes **0.5×**, **VAR-d24** takes **0.6×**, and **VAR-d30** takes **1×**, showing efficient scaling with model size.

Type	Model	FID↓	IS↑	Pre↑	Rec↑	#Para	#Step	Time
GAN	BigGAN [13]	6.95	224.5	**0.89**	0.38	112M	1	–
GAN	GigaGAN [42]	3.45	225.5	0.84	**0.61**	569M	1	–
GAN	StyleGan-XL [74]	2.30	265.1	0.78	0.53	166M	1	0.3 [74]
Diff.	ADM [26]	10.94	101.0	0.69	0.63	554M	250	168 [74]
Diff.	CDM [36]	4.88	158.7	–	–	–	8100	–
Diff.	LDM-4-G [70]	3.60	247.7	–	–	400M	250	–
Diff.	DiT-L/2 [63]	5.02	167.2	0.75	0.57	458M	250	31
Diff.	DiT-XL/2 [63]	2.27	278.2	0.83	0.57	675M	250	45
Diff.	L-DiT-3B [3]	2.10	304.4	0.82	0.60	3.0B	250	>45
Diff.	L-DiT-7B [3]	2.28	316.2	0.83	0.58	7.0B	250	>45
Mask.	MaskGIT [17]	6.18	182.1	0.80	0.51	227M	8	0.5 [17]
Mask.	RCG (cond.) [51]	3.49	215.5	–	–	502M	20	1.9 [51]
AR	VQVAE-2† [68]	31.11	~45	0.36	0.57	13.5B	5120	–
AR	VQGAN† [30]	18.65	80.4	0.78	0.26	227M	256	19 [17]
AR	VQGAN [30]	15.78	74.3	–	–	1.4B	256	24
AR	VQGAN-re [30]	5.20	280.3	–	–	1.4B	256	24
AR	ViTVQ [92]	4.17	175.1	–	–	1.7B	1024	>24
AR	ViTVQ-re [92]	3.04	227.4	–	–	1.7B	1024	>24
AR	RQTran. [50]	7.55	134.0	–	–	3.8B	68	21
AR	RQTran.-re [50]	3.80	323.7	–	–	3.8B	68	21
VAR	VAR-*d16*	3.30	274.4	0.84	0.51	310M	10	0.4
VAR	VAR-*d20*	2.57	302.6	0.83	0.56	600M	10	0.5
VAR	VAR-*d24*	2.09	312.9	0.82	0.59	1.0B	10	0.6
VAR	VAR-*d30*	1.92	323.1	0.82	0.59	2.0B	10	1
VAR	VAR-*d30*-re	**1.73**	**350.2**	0.82	0.60	2.0B	10	1
	(validation data)	1.78	236.9	0.75	0.67			

Does VAR generate better images than prior methods?

Comparison on ImageNet 512×512

- **VAR-d36-s** achieves the **best result** with **FID 2.63** and **IS 303.2**.
- VAR outperforms **DiT-XL/2** (**FID 3.04, IS 240.8**) and clearly surpasses **BigGAN**, **ADM**, **MaskGIT**, and **VQGAN**.
- Inference time for **VAR** is **1**, while **DiT-XL/2** requires **81**, showing a major speed advantage.
- The **"-s"** version uses a **single shared AdaLN layer** because of resource limitations.
- The result shows that **VAR scales effectively to 512×512 resolution** while preserving both **high quality** and **fast generation**.

Type	Model	FID↓	IS↑	Time
GAN	BigGAN [13]	8.43	177.9	–
Diff.	ADM [26]	23.24	101.0	–
Diff.	DiT-XL/2 [63]	3.04	240.8	81
Mask.	MaskGIT [17]	7.32	156.0	0.5†
AR	VQGAN [30]	26.52	66.8	25†
VAR	VAR-*d*36-s	**2.63**	**303.2**	1

Does VAR follow clean scaling laws as model size and compute increase?

- Prior work on **autoregressive large language models** shows that increasing **model size**, **training tokens**, or **training compute** leads to a predictable reduction in **test loss**.
- This relationship follows a **power law**:

$$L = (\beta X)^{\alpha}$$

where X can be **parameter count** N, **training tokens** T, or **optimal training compute** C_{min}.

- After taking logarithms, the relation becomes linear:

$$\log(L) = \alpha \log(X) + \alpha \log(\beta)$$

so a **straight line in log-log scale** indicates scaling-law behavior.

Does VAR follow clean scaling laws as model size and compute increase?

- To test this, they train **12 VAR models** ranging from **18M to 2B parameters**.
- The models are trained on the **ImageNet** training set with **1.28M images**, corresponding to approximately **870B visual tokens per epoch** under the VQ-VAE tokenizer.
- Depending on model size, training lasts **200 to 350 epochs**, with a maximum of **305B training tokens**.

Does VAR follow clean scaling laws as model size and compute increase?

Scaling laws with model parameters N

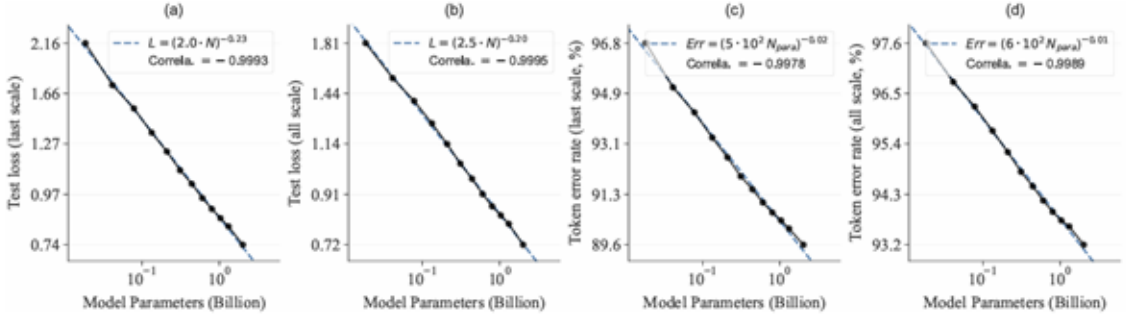


Figure 5: **Scaling laws with VAR transformer size** N, with power-law fits (dashed) and equations (in legend). Small, near-zero exponents α suggest a smooth decline in both test loss L and token error rate Err when scaling up VAR transformer. Axes are all on a logarithmic scale. The Pearson correlation coefficients near -0.998 signify a strong linear relationship between $log(N)$ *vs.* $log(L)$ or $log(N)$ *vs.* $log(Err)$.

- VAR model size is scaled using: $$N(d) = 18d(64d)^2 = 18d \cdot 4096d^2 = 73728d^3.$$
- Depth is varied from **6 to 30**, producing **12 models** from **18.5M to 2.0B parameters**.
- Performance is evaluated using **test loss** and **token error rate** on the **ImageNet validation set**.
- Both **last-scale** and **average-scale** metrics are analyzed.

Does VAR follow clean scaling laws as model size and compute increase?

Scaling Laws with optimal training compute C_min

- he **Pareto frontier** identifies the minimum compute needed to reach a target performance level.
- Both **loss** and **error** follow clear **power-law trends** with respect to C_min.
- The log-log plots are nearly linear, with correlations close to **–0.99**.
- The results show that **larger VAR models are more compute-efficient** when enough training data is available.

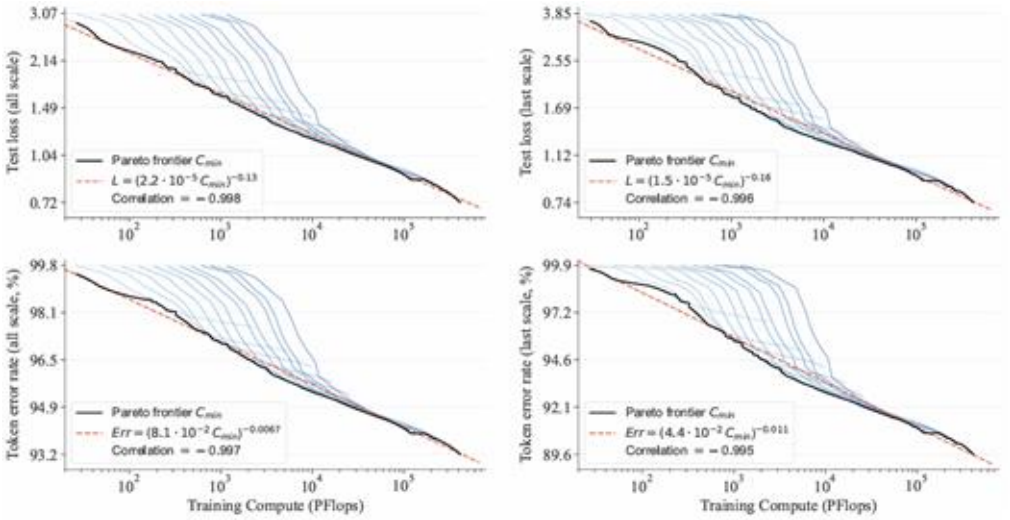


Figure 6: **Scaling laws with optimal training compute C_{min}.** Line color denotes different model sizes. Red dashed lines are power-law fits with equations in legend. Axes are on a logarithmic scale. Pearson coefficients near -0.99 indicate strong linear relationships between $\log(C_{min})$ *vs.* $\log(L)$ or $\log(C_{min})$ *vs.* $\log(Err)$.

Ablation Study of VAR

	Description	Para.	Model	AdaLN	Top-k	CFG	Cost	FID↓	Δ
1	AR [30]	227M	AR	✗	✗	✗	1	18.65	0.00
2	AR to VAR	207M	VAR-$d16$	✗	✗	✗	0.013	5.22	−13.43
3	+AdaLN	310M	VAR-$d16$	✓	✗	✗	0.016	4.95	−13.70
4	+Top-k	310M	VAR-$d16$	✓	600	✗	0.016	4.64	−14.01
5	+CFG	310M	VAR-$d16$	✓	600	2.0	0.022	3.60	−15.05
5	+Attn. Norm.	310M	VAR-$d16$	✓	600	2.0	0.022	3.30	−15.35
6	+Scale up	2.0B	VAR-$d30$	✓	600	2.0	0.052	1.73	−16.85

- **AdaLN**: **Adaptive Layer Normalization**
- **Top-k**: sampling only from the **top-k most likely tokens** during generation
- **CFG**: **Classifier-Free Guidance**
- **Attn. Norm.**: normalizing **query** and **key** vectors before attention
- **Scale up**: increasing the model size to a larger VAR transformer
- **Cost**: inference cost relative to the baseline AR model
- **Δ**: reduction in **FID** compared to the baseline model

Ablation Study of VAR

	Description	Para.	Model	AdaLN	Top-k	CFG	Cost	FID↓	Δ
1	AR [30]	227M	AR	✗	✗	✗	1	18.65	0.00
2	AR to VAR	207M	VAR-$d16$	✗	✗	✗	0.013	5.22	−13.43
3	+AdaLN	310M	VAR-$d16$	✓	✗	✗	0.016	4.95	−13.70
4	+Top-k	310M	VAR-$d16$	✓	600	✗	0.016	4.64	−14.01
5	+CFG	310M	VAR-$d16$	✓	600	2.0	0.022	3.60	−15.05
5	+Attn. Norm.	310M	VAR-$d16$	✓	600	2.0	0.022	3.30	−15.35
6	+Scale up	2.0B	VAR-$d30$	✓	600	2.0	0.052	1.73	−16.85

- Baseline **AR** model: **FID 18.65**, cost **1.0**
- Replacing AR with **VAR** gives a major gain: **FID 5.22**, cost **0.013**

Ablation Study of VAR

	Description	Para.	Model	AdaLN	Top-k	CFG	Cost	FID↓	Δ
1	AR [30]	227M	AR	✗	✗	✗	1	18.65	0.00
2	AR to VAR	207M	VAR-$d16$	✗	✗	✗	0.013	5.22	−13.43
3	+AdaLN	310M	VAR-$d16$	✓	✗	✗	0.016	4.95	−13.70
4	+Top-k	310M	VAR-$d16$	✓	600	✗	0.016	4.64	−14.01
5	+CFG	310M	VAR-$d16$	✓	600	2.0	0.022	3.60	−15.05
5	+Attn. Norm.	310M	VAR-$d16$	✓	600	2.0	0.022	3.30	−15.35
6	+Scale up	2.0B	VAR-$d30$	✓	600	2.0	0.052	1.73	−16.85

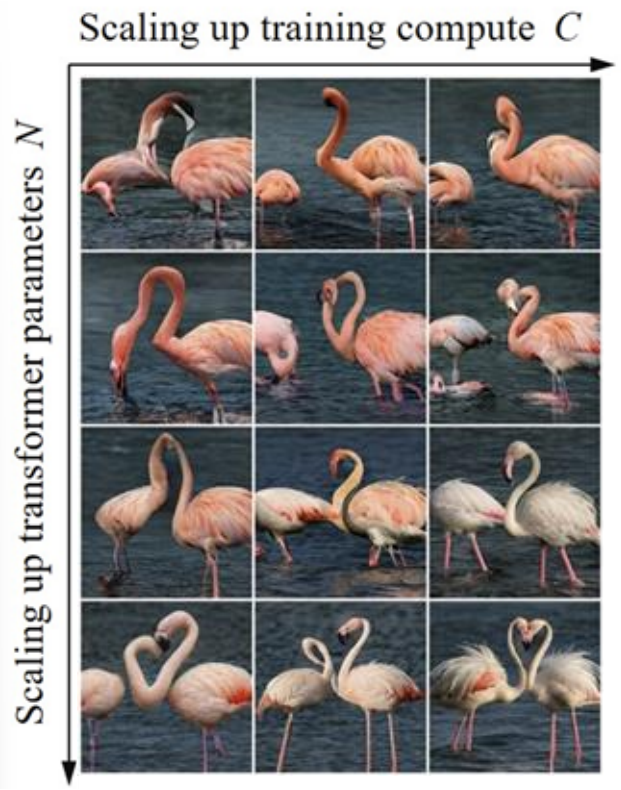
- **AdaLN** improves FID to **4.95**
- **Top-k sampling** improves FID to **4.64**
- **CFG** improves FID to **3.60**
- **Attention normalization** improves FID to **3.30**

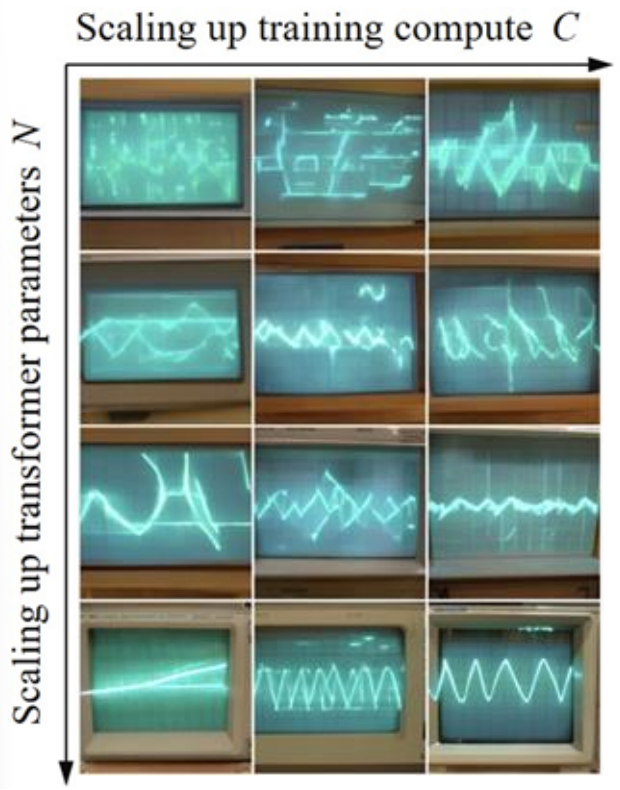
Ablation Study of VAR

	Description	Para.	Model	AdaLN	Top-k	CFG	Cost	FID↓	Δ
1	AR [30]	227M	AR	✗	✗	✗	1	18.65	0.00
2	AR to VAR	207M	VAR-$d16$	✗	✗	✗	0.013	5.22	−13.43
3	+AdaLN	310M	VAR-$d16$	✓	✗	✗	0.016	4.95	−13.70
4	+Top-k	310M	VAR-$d16$	✓	600	✗	0.016	4.64	−14.01
5	+CFG	310M	VAR-$d16$	✓	600	2.0	0.022	3.60	−15.05
5	+Attn. Norm.	310M	VAR-$d16$	✓	600	2.0	0.022	3.30	−15.35
6	+Scale up	2.0B	VAR-$d30$	✓	600	2.0	0.052	1.73	−16.85

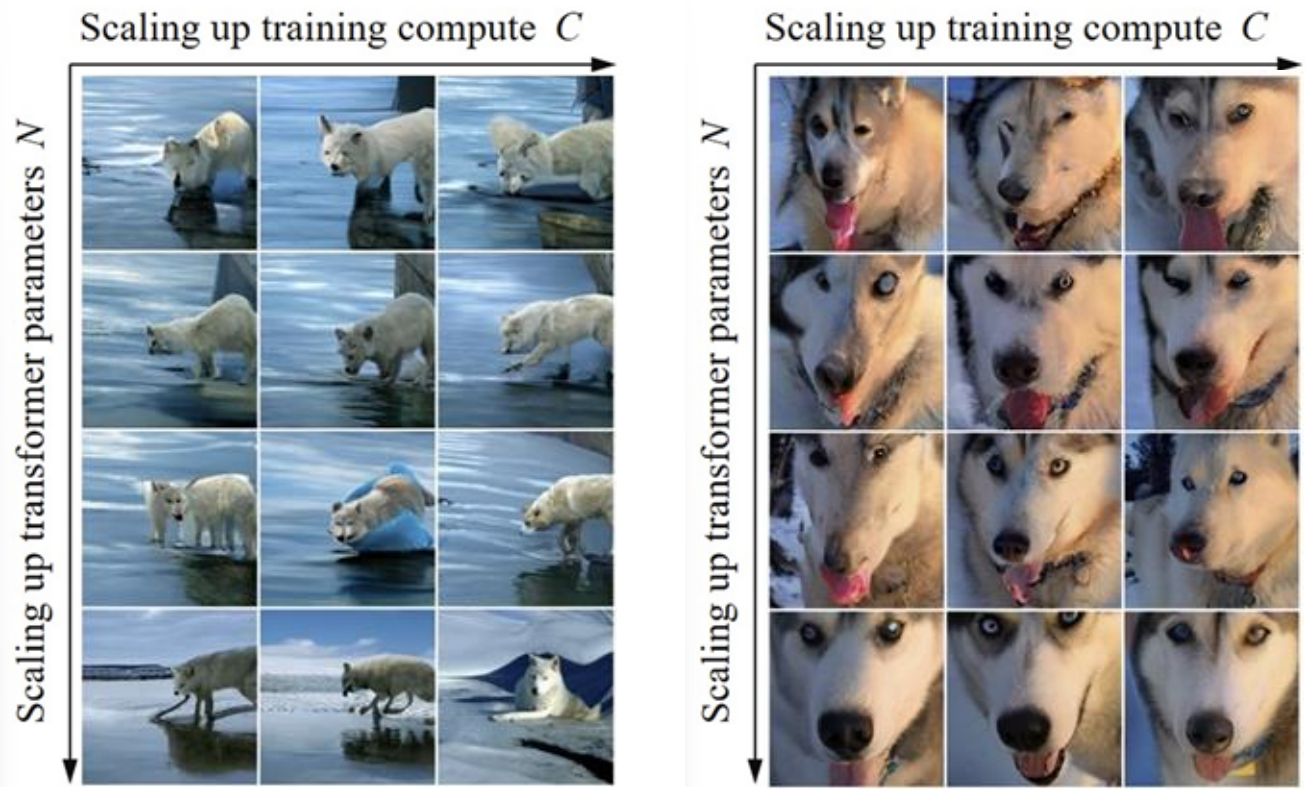
- Scaling up to **VAR-d30 (2.0B)** achieves **FID 1.73**
- Final improvement over baseline: **−16.85 FID**
- Key message: **the main gain comes from the VAR framework, with further improvements from architectural and sampling enhancements**

What does scaling look like qualitatively in generated images?

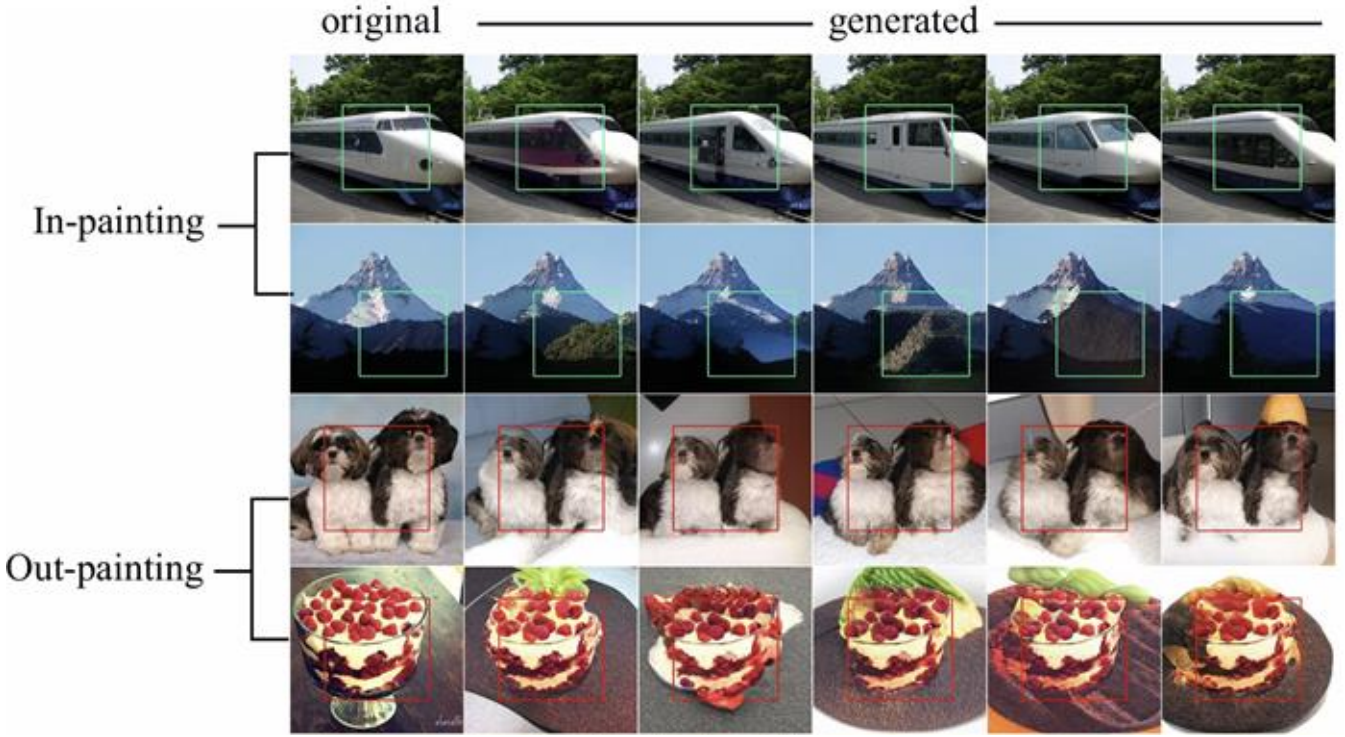




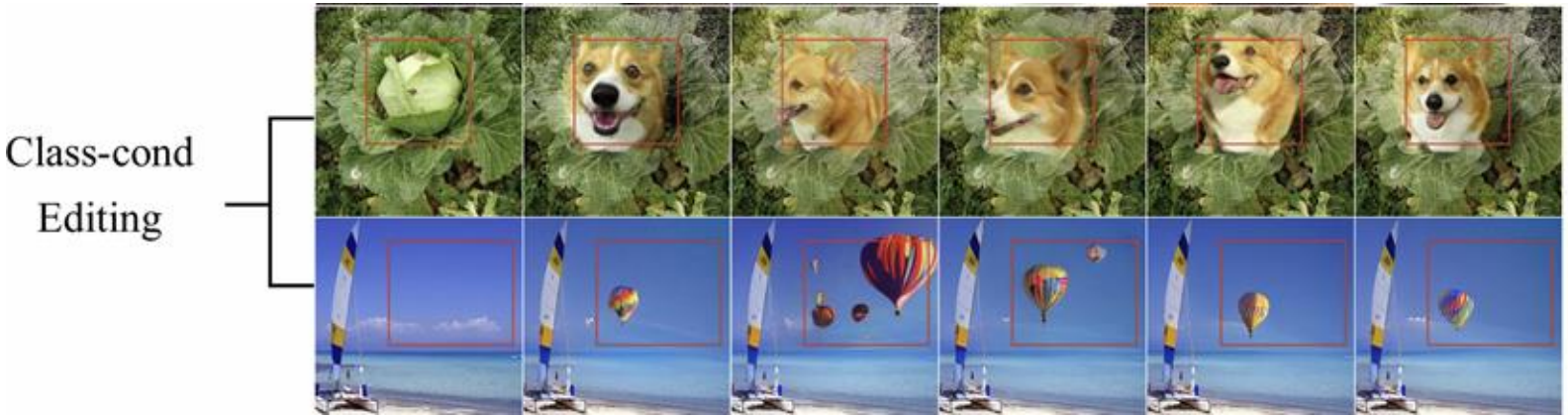
What does scaling look like qualitatively in generated images?



Zero-shot task generalization



Zero-shot task generalization



Future Work and limitation

Limitations

- VQVAE tokenizer architecture and training kept unchanged from baseline to isolate VAR's contribution
- Text-prompt generation not implemented — only class-conditional generation evaluated
- Video generation not implemented, though the authors discuss a natural "3D next-scale prediction" extension

Future Directions

- Advance VQVAE tokenizer (e.g. FSQ, MoVQ) — orthogonal improvement to VAR framework
- Text-to-image via LLM integration (encoder-decoder or in-context)

→ *Realized: Infinity (CVPR 2025 Oral)*

- Video generation via "3D next-scale prediction" over spatiotemporal pyramids

→ *Realized: InfinityStar (NeurIPS 2025 Oral)*

Take home message

- Choosing a tokenization method to respect:
  - Unidirectional probabilistic principal of autoregressive methods
  - Locality and position in data

  is key for maximizing performance and attaining the benefits of transformers (scaling and zero shot generalization)
- For images, this paper shows that next scale token generation is a much more appropriate method that fits these assumptions better

THE UNIVERSITY OF BRITISH COLUMBIA