RandAR: Decoder-only Autoregressive Visual Generation in Random Orders

Supplementary Material

In this supplementary material, we first provide the pseudo-code (Sec. A) and additional implementation details (Sec. B). Then, we provide analyses on different generation orders (Sec. C) and results at intermediate training steps (Sec. D) and 384×384 resolution generation performance (Sec. E). We conduct a more in-depth analysis of bi-directional feature encoding (Sec. F) and spatial contextual guidance (Sec. G). We finally present additional visualization of the generated images (Sec. H) and discuss the limitations and future works (Sec. I).

A. Pseudo-Code

We have provided the Pytorch-style pseudo-code for our random-order training (Algorithm A) and parallel decoding inference (Algorithm B). We will release the code upon acceptance.

B. Additional Implementation Details

For image preprocessing, we follow the approach in LlamaGen [44]. Specifically, for 256×256 experiments, we resize the image's shorter edge to 256×1.1 and apply a center square crop with the same size, and then perform ten-crop augmentation.

All the experiments in Table 1, including raster-order counterparts, use classifier-free guidance (CFG) with a linear schedule for sampling. The optimal CFG weight is determined through a sweep with a step size of 0.1 across all methods. We use a temperature of 1.0 without applying top-k filtering.

Model Configurations. We have provided several sizes of the RandAR model following LLaMAGen [44], which are plain decoder-only transformers. The detailed architectures are in Table A.

Table A. Model configurations of RandAR.

Model	Parameters	Layers	Hidden Dim	Attn Heads
RandAR-L	343M	24	1024	16
RandAR-XL	775M	36	1280	20
RandAR-XXL	1.4B	48	1536	24

C. Ablation Study on Generation Orders

Generation Orders. In Table B, we analyze RandAR with different generation orders from VQGAN [10]: spiral-in, spiral-out, z-curve, subsample, and alternate. These orders are visualized in Fig. A using 4×4 grids.

Inference-time Orders. RandAR can utilize arbitrary generation orders by training on random generation orders. In addition to the row-major raster order, we investigate the generation orders as explained above. Because of the random-order generation ability of RandAR, we further propose some prior knowledge for using *partially* random orders: (1) Hierarchical, which is used for our resolution extrapolation and first generates the tokens at even coordinates for a global layout; (2) Center-first, which first generates the tokens at the center 1/2×1/2 of the image; (3) Border-first, which first generates the background before the center tokens. The visualization of these orders is in Fig. A.

Table B. Ablation studies on *generation orders* for RandAR. We experiment with the default fully-randomized orders, the partially randomized orders guided by priors, and the fixed orders explored in VQGAN [10]. We discover that the default random order performs the best, and the orders of "hierarchical random" and "subsample" also outperform other orders. These indicate that RandAR benefits from the overall image contexts provided by more divergent token locations at initial steps.

Order	FID↓	IS↑	Precision↑	Recall↑	Steps
Random	2.25	317.8	0.80	0.60	88
Hierarchical Random	2.36	310.2	0.80	0.60	88
Center-first Random	2.97	262.8	0.76	0.63	88
Border-first Random	2.56	300.5	0.78	0.61	88
Raster	4.82	299.2	0.71	0.60	256
Spiral-in	3.36	280.1	0.72	0.64	256
Spiral-out	3.79	239.5	0.73	0.65	256
Z-curve	4.00	317.3	0.73	0.60	256
Subsample	2.40	298.8	0.79	0.61	256
Alternate	4.29	307.2	0.72	0.58	256

The experimental results are shown in Table B. It indicates that a fully randomized order, the default choice of RandAR, performs the best. We conjecture that a fully randomized order best leverages the RandAR's capability of combining contexts from different locations of the images. Interestingly, the generation orders that encourage more divergent token locations at initial steps, *i.e.*, hierarchical random and subsample, perform better than the other orders, potentially via covering a larger range of image contexts.

D. Performance at Middle Training Steps

We provide the additional results of our RandAR models at intermediate training steps in Table C. All the results are evaluated with 88 steps of generation with parallel decoding described in Sec. 3.3.

E. RandAR on ImageNet at 384 Resolution

We report results on ImageNet at 384×384 resolution using the same tokenizer, which produces 24×24 tokens per image, corresponding to 576! random permutations. Our XL-sized model, with 775M parameters, is trained using the

Algorithm A RandAR Training Pytorch-style Pseudo-Code.

```
# Random order training.
# Input list:
# class_indices: [b, 1], b is batch size, dtype of torch.long;
# b, h, w: int, batch_size, height and width for latent space size
# img_token_indices: [b, h * w], image token indices after the tokenizer
# d: the hidden dimension of the model
# head_dim: dimension of each attention head
# model: the decoder-only transformer
# Output: training loss

# Step-1: Sample random orders
seq_len = h * w
raster_order_indices = torch.arange(seq_len).repeat(b, 1) # [b, seq_len]
position_indices = random_permute(raster_order_indices) # [b, seq_len]

# Step-2: Prepare embeddings
image_tokens = model.token_embeddings[image_token_indices] # [b, seq_len, d]
image_tokens = torch.gather(image_tokens.unsqueeze(-1), dim=1, position_indices.unsqueeze(-1))
cls_token = model.cls_embeddings[class_indices] # [b, d]

# Random dropout
image_tokens = random_dropout(image_tokens, p=0.1)
cls_token = random_dropout(cls_token, p=0.1)

# Step-3: Compute position instructions tokens
# get 2D RoPE frequencies for each spatial location
rope_freqs_cis = model.compute_rope_frequencies(b, h, w, base=10000) # [b, h, w, head_dim//2, 2]
# flatten h, w to seq_len, arranging 2D RoPE frequencies in raster order
rope_freqs_cis = rope_freqs_cis.flatten((1, 2)) # [b, seq_len, head_dim//2, 2]
# get 2D rope frequencies in permuted random orders
rope_freqs_cis = rope_freqs_cis[position_indices]

# get position instruction tokens corresponding to tokens in random order
pos_instruct_tokens = apply_2d_rope(model.shared_pos_embed, rope_freqs_cis) # [b, seq_len, d]

# Step 4: Prepare Teacher Forcing Sequences
x = torch.zeros(b, 1 + 2 * seq_len, d).to(image_tokens.device)
x[:, 0] = cls_token
x[:, 1::2] = pos_instruct_tokens
x[:, 2::2] = image_tokens

x_rope_freqs = torch.zeros(b, 1 + 2 * seq_len, head_dim // 2)
x_rope_freqs[:, 0] = model.class_rope_freqs
x_rope_freqs[:, 1::2] = rope_freqs_cis # rope for position instruction tokens
x_rope_freqs[:, 2::2] = rope_freqs_cis # rope for image tokens

# Step-5: Training with Next-token Prediction
pred_logits = model(x, x_rope_freqs) # [b, 1 + 2 * seq_len, vocab_size]

# generated tokens from position instruction tokens
pred_logits = pred_logits[:, 1::2] # [b, seq_len, vocab_size]

# return back to raster order sequence for loss computation
index_to_raster_order = torch.argsort(position_indices) # [bs, seq_len]
raster_pred_logits = torch.gather(pred_logits, dim=1, index_to_raster_order.unsqueeze(-1))

loss = cross_entropy(raster_pred_logits.view(-1, vocab_size), image_token_indices.view(-1))
return loss
```

same setup. With 180 sampling steps, the model achieves an FID of 2.32 and an Inception Score of 323. Using 144 sampling steps, it achieves an FID of 2.35 and an Inception Score of 322. The FID is slightly higher than that of the 256×256 model, consistent with observations in LlamaGen [44] that models smaller than 1B parameters perform slightly worse at 384×384 resolution.

F. Additional Results on Feature Encoding

We have conducted feature encoding experiments in Sec. 4.4.4 and Table 4, suggesting that decoder-only transformers learned in random generation orders can generalize to extracting features from bi-directional contexts, while raster-order models cannot. In this section, we provide additional ablation studies and discussions.

Comparison with VQ Tokenizer. Here we demonstrate that our autoregressive transformer learns better representation than its VQ tokenizer, which provides the input token indices to our RandAR transformers. Specifically, we conduct the feature correspondence experiment on SPari71k [32] with the DIFT [45] framework, only replacing the feature extractor with the encoder from the VQ Tokenizer. As shown in Table D, VQ Tokenizer performs significantly worse than our RandAR.

Transformer Layers in RandAR. As noticed by previous work [33, 63], varied layers from a decoder-only language model can have significantly different abilities for vi-

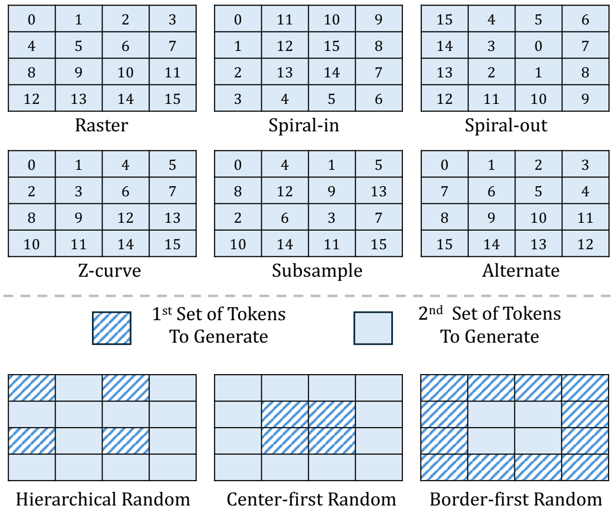

Figure A. Illustration of different generation orders. RandAR, by default, uses a fully randomized order for inference. We investigate the fixed generation orders proposed in VQGAN [10] (top) and partially random orders guided by the priors of hierarchy, center-first, and border-first (bottom).

Table C. Generation Results for Intermediate Training Steps. With a batch size of 1024, 300 epochs of full RandAR training equals 360 iterations.

Model	Iters	FID↓	IS↑	Precision↑	Recall↑
RandAR-L	50k	4.21	224.2	0.83	0.49
	100k	3.85	251.4	0.82	0.52
	300k	3.21	259.7	0.80	0.55
RandAR-XL	50k	3.11	271.1	0.81	0.53
	100k	2.82	293.6	0.81	0.56
	300k	2.66	296.3	0.80	0.57
RandAR-XXL	50k	3.01	277.4	0.79	0.57
	100k	2.61	296.5	0.79	0.57
	300k	2.37	309.5	0.79	0.60

sual feature encoding. RandAR has a similar case since the earlier layers might concentrate on low-level patterns while later layers are primarily used to map the features into the token space. For our 775M model, which has 36 layers, we analyze the performance difference for 12-th, 24-th, and 36-th layers. As shown in Table D, the 24-th layer performs the best for both random and raster order models; thus, it is used for our comparison in Table 4 in the main paper.

G. Spatial Contextual Guidance

In Sec. 3.4.3, we introduce a new type of guidance called "*Spatial Contextual Guidance*" (SCG) inspired by the classifier-free guidance (CFG). This section describes SCG in detail and analyzes its benefits.

G.1. Formulation of Spatial Contextual Guidance

The motivation of SCG is to enable better consistency in high-frequency details, as shown in Fig. 6. Inspired by CFG, SCG guides the generation by calculating the differ-

Table D. Ablation Studies for Finding Feature Correspondences on SPair71k [32]. We search for the best decoder-only transformer learning for feature encoding (24-th), and then show that the features from our RandAR transformer are better than those from the VQ image tokenizer.

Model	Layer	Feature Correspondence (SPair71k) PCK (Per Image) ↑	PCK (Per Point) ↑
RasterAR	24-th	24.5	28.6
w/ 2nd Round		3.6	3.9
RandAR		22.1	25.8
w/ 2nd Round		**31.3**	**36.4**
VQ Tokenizer	-	5.6	6.0
RasterAR	12-th	10.7	12.4
RasterAR w/ 2nd Round		3.5	3.7
RandAR w/ 2nd Round		16.3	19.0
RandAR		11.0	12.6
RasterAR	36-th	11.1	12.5
RasterAR w/ 2nd Round		3.5	3.7
RandAR		2.4	2.5
RandAR w/ 2nd Round		10.3	11.2

ence between the two sampling results with *all the previous tokens* as context and *part of the previous tokens as context*. Denoting the RanAR network as $e_\theta(\cdot)$, the spatial contextual guidance is:

$$\tilde{e}_\theta(\mathbf{x}_{1:n}, c) = e_\theta(\mathbf{x}^\phi_{1:n}, c) + w_{\text{scg}}(e_\theta(\mathbf{x}_{1:n}, c) - (e_\theta(\mathbf{x}^\phi_{1:n}, c)), \tag{A}$$

where c is the class conditioning, $\mathbf{x}_{1:n}$ is the set of tokens generated in previous steps, and $\mathbf{x}^\phi_{1:n}$ is the set of tokens with a random dropout. With such guidance, the final generated result $\tilde{e}_\theta(\mathbf{x}_{1:n}, c)$ has better consistency with the tokens dropped out from $\mathbf{x}_{1:n}$.

When combining SCG with the conventional CFG, we follow InstructPix2Pix [2] and Liu *et al.* [26] to compose two guidances together:

$$\begin{aligned} \tilde{e}_\theta(\mathbf{x}_{1:n}, c) = & e_\theta(\mathbf{x}^\phi_{1:n}, c^\phi) + \\ & w_{\text{scg}}(e_\theta(\mathbf{x}_{1:n}, c^\phi) - e_\theta(\mathbf{x}^\phi_{1:n}, c^\phi)) + \\ & w_{\text{cfg}}(e_\theta(\mathbf{x}_{1:n}, c) - e_\theta(\mathbf{x}_{1:n}, c^\phi)), \end{aligned} \tag{B}$$

where $w_{\text{scg}} = 1$ will make the above guidance equivalent to conventional CFG.

SCG is supported by the training since the image tokens experience a random dropout of 10%, following the standard practice in LLaMAGen [44]. During the inference time, we randomly dropout a token to all zeros by the probability of 25% to create $\mathbf{x}^\phi_{1:n}$. In this way, the generation process is still fully autoregressive and compatible with KV-cache.

G.2. Evaluation of Spatial Contextual Guidance

SCG can improve the visual quality of regular resolution generation and resolution extrapolation.

Resolution Extrapolation. As shown in Fig. B, using SCG enhances the high-frequency details of the images when generating 512×512 images directly from our 775M RandAR trained from 256×256. Numerous extraneous

Figure B. Using spatial contextual guidance (SCG) significantly improves the visual quality for zero-shot resolution extrapolation, especially the high-frequency details. The images are all 512×512, and the "w/ SCG" samples are from $w_{\text{scg}} = 2.5$. (Zoom in for high-resolution details.)

Table E. Ablation Study of Spatial Contextual Guidance (SCG). SCG can improve the visual quality for random-order AR models, where it decreases sFID by a large margin with a minor drop in FID. Although SCG is only designed for resolution extrapolation, it implicitly reflects the advantage of RandAR in combining bi-directional context.

Model	SCG	FID↓	sFID↓	IS↑	Precision↑	Recall↑
RandAR	✗	2.25	6.13	317.8	0.80	0.60
	✓	2.34	5.85	303.8	0.80	0.60

parts of the objects and uneven patterns are removed.

Regular 256×256 Generation. Although SCG is primarily proposed for the challenging zero-shot resolution extrapolation, its effects are also reflected in regular 256×256 generation. Our observation is also validated by quantitative evaluation. As in Table E evaluating the 775M RandAR model, SCG of $w_{\text{scg}} = 1.2$ can improve sFID, which emphasizes more low-level details, at a marginal drop of FID.

H. Additional Generation Results

H.1. Outpainting

We provide additional visualizations of the outpainting results in Fig. C.

H.2. Regular Image Generation

We demonstrate the uncurated 256×256 images generated from our 775M RandAR-XL. They are displayed from Fig. D to Fig. I.

H.3. Resolution Extrapolation

We provide uncurated resolution extrapolation results from Fig. J to Fig. O with 775M RandAR-XL. Our zero-shot extrapolation produces high-quality images with unified layouts and detailed patterns like furs of dogs (Fig. J), coral reefs (Fig. K), and scenery (Fig. M). However, we also notice that zero-shot resolution extrapolation is a challenging task. As the model has never been trained on high-frequency details, it will struggle with the small patterns, *e.g.*, eyes and noses of dogs (Fig. J, Fig. L) and straight shapes of man-made objects (Fig. N).

Figure C. 4× Outpainting results using 256×256 RandAR to generate 256×1024 images. Full sequence attention is used.

I. Limitations and Future Works

Our RandAR investigates enabling decoder-only transformers to generate image tokens in random orders. Although it illustrates the advantages of combining bi-directional contexts from images, random-order generation so far achieves comparable performance compared with the raster-order counterparts, as learning from a much larger number of orders is significantly more challenging. Therefore, a meaningful future investigation would be improving the data efficiency of training a random-order model.

In addition, we notice the trend of joint visual language generation with decoder-only transformers [46, 51, 62], which uniformly follows a raster-order design. From this aspect, RandAR can be further scaled up from ImageNet pre-training to the image-text and image-video datasets.

Algorithm B RandAR Parallel Decoding Pytorch-style Pseudo-Code.

```
# Parallel decoding with cosine step size schedule. Classifier-free guidance is omitted for simplicity.
# Input list:
# class_indices: [b, 1], b is batch size, dtype of torch.long;
# b, h, w: int, batch_size, height and width for latent space size
# d: the hidden dimension of the model
# model, vq_vae: the decoder-only transformer and the Vector quantized VAE
# Output: a batch of generated images

# Step-1: Sample random orders
seq_len = h * w
raster_order_indices = torch.arange(seq_len).repeat(b, 1) # [b, seq_len]
position_indices = random_permute(raster_order_indices) # [b, seq_len]

# Step-2: Compute position instructions tokens
# get 2D RoPE frequencies for each spatial location
rope_freqs_cis = model.compute_rope_frequencies(b, h, w, base=10000) # [b, h, w, head_dim//2, 2]
# flatten h, w to seq_len, arranging 2D RoPE frequencies in raster order
rope_freqs_cis = rope_freqs_cis.flatten((1, 2)) # [b, seq_len, head_dim//2, 2]
# get 2D rope frequencies in permuted random orders
rope_freqs_cis = rope_freqs_cis[position_indices]
# get position instruction tokens corresponding to tokens in random order
pos_instruct_tokens = apply_2d_rope(model.shared_pos_embed, rope_freqs_cis) # [b, seq_len, d]

# Step-3: Init KV-caches & Class_embedding & PlaceHolder for generated tokens
max_token_length = 1 + seq_len * 2 # class_embedding + position instruction tokens + image tokens
model.setup_KVcache(max_token_length, batch_size=b)
class_embed = model.class_embedding(class_indices) # [b, 1, d]
generated_code_indices = torch.zeros((b, seq_len), dtype=torch.long) # [b, seq_len], placeholder
num_generated = 0

# Step-4: Prefill: prepare input of first decoding iteration
step_size = 1 # number of decoding tokens for next iteration. starting at one-token-each-time
x = torch.cat([class_embed, pos_instruct_embeddings[:, 0:1]], dim=1) # [b, 2, d]
x_rope_freqs = torch.cat([model.class_rope_freqs, rope_freqs_cis[:, 0:1]], dim=1) # [b, 2, head_dim//2, 2]
kvcache_write_indices = torch.arange(2)

# Step-5: Start decoding loop. Using Parallel decoding with Cosine step-size schedule
while num_generated < seq_len:
   pred_logits = model(x, x_rope_freqs, kvcache_write_indices) # [b, num_cur_tokens, vocab_size]
   pred_logits = pred_logits[:, -step_size:] # [b, num_query_tokens, vocab_size]

   sampled_indices = sample(pred_logits, temperature=1.0, topk=-1) # [b, step_size] in torch.long
   generated_code_indices[:, num_generated:num_generated+step_size] = sampled_indices
   sampled_tokens = model.token_embedding(sampled_indices) # [b, step_size, d]

   # prepare input x, x_rope_freqs, kvcache_write_indices for next iterations.
   step_size_next = CosineSchedule(num_generated, seq_len)
   # suppose the step size of last iteration is 2, the model decoded two image tokens, denoting as i1, i2.
   # denote the position instruction token for these two decoded tokens as p1, p2.
   # Then in last iteration, the input tensor x is [...., p1, p2].

   # suppose the step size for the next iteration remains as 2,
   # with new position instruction tokens p3 and p4,
   # then the input tensor x for the next iteration would be [i1, p2, i2, p3, p4].
   # Note We rewrite the KV-cache corresponding to [i1, p2, i2],
   # so that the effective KV-cache follows the interleave format: [..., p1, i1, p2, i2, ...],
   # consistent with training format.

   # the number of input tokens for next iteration would be: 2 * step_size + step_size_next -1
   x = torch.zeros((b, 2 * step_size + step_size_next -1, d))
   x_rope_freqs = torch.zeros((b, 2 * step_size + step_size_next -1, head_dim // 2, 2))
   kvcache_write_indices = torch.arange(2 * step_size + step_size_next -1) + kvcache_write_indices[1-
        step_size]
   # using examples in above comments, fill in the [i1, p2, i2] part
   x[:, 0] = sampled_tokens[:, 1]
   x_rope_freqs[:, 0] = rope_freqs_cis[:, num_generated]
   for i in range(step_size - 1):
      x[:, 2 * i + 1] = pos_instruct_tokens[:, num_generated + i + 1]
      x[:, 2 * i + 2] = sampled_tokens[:, i + 1]
      x_rope_freqs[:, 2 * i + 1] = rope_freqs_cis[:, num_generated + i + 1]
      x_rope_freqs[:, 2 * i + 2] = rope_freqs_cis[:, num_generated + i + 1]

   num_generated += step_size
   step_size = step_size_next
   # using examples in above comments, fill in the [p3, p4] part
   x[:, -step_size_next:] = pos_instruct_tokens[:, num_generated:num_generated + step_size_next]
   x_rope_freqs[:, -step_size_next:] = rope_freqs_cis[:, num_generated:num_generated + step_size_next]

# Step-6, decode generated tokens to images
index_to_raster_order = torch.argsort(position_indices) # [bs, seq_len, 1]
generated_code_indices = torch.gather(generated_code_indices.unsqueeze(-1), dim=1, index_to_raster_order)
img = vq_vae.decode(generated_code_indices)
```

Figure D. Uncurated generation results (256×256). $w_{\text{cfg}} = 4.0$. Golden retriever (ImageNet class 207).

Figure E. Uncurated generation results (256×256). $w_{\text{cfg}} = 4.0$. Husky (ImageNet class 250).

Figure F. Uncurated generation results (256×256). $w_{\text{cfg}} = 4.0$. Balloon (ImageNet class 417).

Figure G. Uncurated generation results (256×256). $w_{\text{cfg}} = 4.0$. Volcano (ImageNet class 980).

Figure H. Uncurated Generation Results (256×256). $w_{\text{cfg}} = 4.0$. Schooner (ImageNet class 780).

Figure I. Uncurated Generation Results (256×256). $w_{\text{cfg}} = 4.0$. Space shuttle (ImageNet class 812).

Figure J. Uncurated **Zero-shot Resolution Extrapolation**. $w_{\text{cfg}} = 3.0$, $w_{\text{scg}} = 2.5$. Golden retriever (ImageNet class 207).

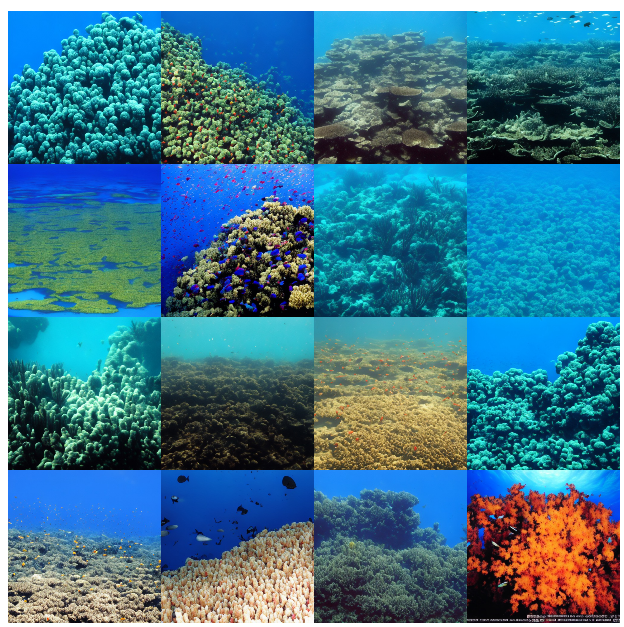

Figure K. Uncurated **Zero-shot Resolution Extrapolation**. $w_{\text{cfg}} = 3.0$, $w_{\text{scg}} = 2.5$. Coral reef (ImageNet class 973).

Figure L. Uncurated **Zero-shot Resolution Extrapolation**. $w_{\text{cfg}} = 3.0$, $w_{\text{scg}} = 2.5$. Husky (ImageNet class 250).

Figure M. Uncurated **Zero-shot Resolution Extrapolation**. $w_{\text{cfg}} = 3.0$, $w_{\text{scg}} = 2.5$. Volcano (ImageNet class 980).

Figure N. Uncurated **Zero-shot Resolution Extrapolation**. $w_{\text{cfg}} = 3.0$, $w_{\text{scg}} = 2.5$. Lighthouse (ImageNet class 437).

Figure O. Uncurated **Zero-shot Resolution Extrapolation**. $w_{\text{cfg}} = 3.0$, $w_{\text{scg}} = 2.5$. Schooner (ImageNet class 780).

References

[1] Andrew Brock. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 6

[2] Tim Brooks, Aleksander Holynski, and Alexei A Efros. InstructPix2Pix: Learning to follow image editing instructions. In *CVPR*, 2023. 3

[3] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 2, 3

[4] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *ICML*, 2024. 4

[5] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *CVPR*, 2022. 1, 2, 4, 6, 7, 8

[6] Huiwen Chang, Han Zhang, Jarred Barber, AJ Maschinot, José Lezama, Lu Jiang, Ming-Hsuan Yang, Kevin Murphy, William T Freeman, Michael Rubinstein, et al. Muse: Text-to-image generation via masked generative transformers. In *ICML*, 2023. 2

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 2, 4, 5, 8

[8] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In *NeurIPS*, 2021. 6

[9] Zheng Ding, Mengqi Zhang, Jiajun Wu, and Zhuowen Tu. Patched denoising diffusion models for high-resolution image synthesis. In *ICLR*, 2023. 8

[10] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021. 2, 3, 4, 5, 6, 7, 1

[11] Lijie Fan, Tianhong Li, Siyang Qin, Yuanzhen Li, Chen Sun, Michael Rubinstein, Deqing Sun, Kaiming He, and Yonglong Tian. Fluid: Scaling autoregressive text-to-image generative models with continuous tokens. *arXiv preprint arXiv:2410.13863*, 2024. 2

[12] Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. Data engineering for scaling language models to 128k context. In *ICML*, 2024. 5

[13] Shanghua Gao, Pan Zhou, Ming-Ming Cheng, and Shuicheng Yan. Masked diffusion transformer is a strong image synthesizer. In *ICCV*, 2023. 2

[14] Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. Olmo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024. 2

[15] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022. 8

[16] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017. 6

[17] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. 5, 6

[18] Minguk Kang, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. Scaling up gans for text-to-image synthesis. In *CVPR*, 2023. 6

[19] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019. 2

[20] Sehoon Kim, Coleman Richard Charles Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-sparse quantization. In *ICML*, 2024. 4

[21] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[22] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In *NeurIPS*, 2019. 6

[23] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *CVPR*, 2022. 2, 6

[24] Tianhong Li, Huiwen Chang, Shlok Mishra, Han Zhang, Dina Katabi, and Dilip Krishnan. Mage: Masked generative encoder to unify representation learning and image synthesis. In *CVPR*, 2023. 2

[25] Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. In *NeurIPS*, 2024. 1, 2, 3, 4, 6, 7

[26] Nan Liu, Shuang Li, Yilun Du, Antonio Torralba, and Joshua B Tenenbaum. Compositional visual generation with composable diffusion models. In *ECCV*, 2022. 3

[27] Wenze Liu, Le Zhuo, Yi Xin, Sheng Xia, Peng Gao, and Xiangyu Yue. Customize your visual autoregressive recipe with set autoregressive modeling. *arXiv preprint arXiv:2410.10511*, 2024. 2, 6

[28] Yinhan Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364, 2019. 2

[29] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 6

[30] Zhuoyan Luo, Fengyuan Shi, Yixiao Ge, Yujiu Yang, Limin Wang, and Ying Shan. Open-magvit2: An open-source project toward democratizing auto-regressive visual generation. *arXiv preprint arXiv:2409.04410*, 2024. 2, 6

[31] Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. SIT: Exploring flow and diffusion-based generative models with scalable interpolant transformers. *arXiv preprint arXiv:2401.08740*, 2024. 6

[32] Juhong Min, Jongmin Lee, Jean Ponce, and Minsu Cho. Spair-71k: A large-scale benchmark for semantic correspondence. *arXiv preprint arXiv:1908.10543*, 2019. 8, 2, 3

[33] Ziqi Pang, Ziyang Xie, Yunze Man, and Yu-Xiong Wang. Frozen transformers in language models are effective visual encoder layers. In *ICLR*, 2024. 2

[34] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023. 6

[35] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. *Technical report, OpenAI*, 2018. 2

[36] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020. 2

[37] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *ICML*, 2021. 2

[38] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 6

[39] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NeurIPS*, 2016. 6

[40] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. In *SIGGRAPH*, 2022. 6

[41] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019. 4

[42] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020. 5

[43] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. 3, 4, 5, 6, 7

[44] Peize Sun, Yi Jiang, Shoufa Chen, Shilong Zhang, Bingyue Peng, Ping Luo, and Zehuan Yuan. Autoregressive model beats diffusion: Llama for scalable image generation. *arXiv preprint arXiv:2406.06525*, 2024. 1, 2, 3, 4, 5, 6, 7

[45] Luming Tang, Menglin Jia, Qianqian Wang, Cheng Perng Phoo, and Bharath Hariharan. Emergent correspondence from image diffusion. In *NeurIPS*, 2023. 8, 2

[46] Chameleon Team. Chameleon: Mixed-modal early-fusion foundation models. *arXiv preprint arXiv:2405.09818*, 2024. 1, 2, 4

[47] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. In *NeurIPS*, 2024. 2, 6, 7

[48] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 2, 5

[49] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 2

[50] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In *NeurIPS*, 2016. 3

[51] Xinlong Wang, Xiaosong Zhang, Zhengxiong Luo, Quan Sun, Yufeng Cui, Jinsheng Wang, Fan Zhang, Yueze Wang, Zhen Li, Qiying Yu, Yingli Zhao, Yulong Ao, Xuebin Min, Tao Li, Boya Wu, Bo Zhao, Bowen Zhang, Liangdong Wang, Guang Liu, Zheqi He, Xi Yang, Jingjing Liu, Yonghua Lin, Tiejun Huang, and Zhongyuan Wang. Emu3: Next-token prediction is all you need. *arXiv preprint arXiv:2409.18869*, 2024. 1, 2, 4

[52] Yuqing Wang, Shuhuai Ren, Zhijie Lin, Yujin Han, Haoyuan Guo, Zhenheng Yang, Difan Zou, Jiashi Feng, and Xihui Liu. Parallelized autoregressive visual generation. *arXiv preprint arXiv:2412.15119*, 2024. 2

[53] Mark Weber, Lijun Yu, Qihang Yu, Xueqing Deng, Xiaohui Shen, Daniel Cremers, and Liang-Chieh Chen. Maskbit: Embedding-free image generation via bit tokens. *arXiv preprint arXiv:2409.16211*, 2024. 2

[54] Jinheng Xie, Weijia Mao, Zechen Bai, David Junhao Zhang, Weihao Wang, Kevin Qinghong Lin, Yuchao Gu, Zhijie Chen, Zhenheng Yang, and Mike Zheng Shou. Show-o: One single transformer to unify multimodal understanding and generation. *arXiv preprint arXiv:2408.12528*, 2024. 2

[55] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019. 2, 3

[56] Lijun Yu, Yong Cheng, Kihyuk Sohn, José Lezama, Han Zhang, Huiwen Chang, Alexander G. Hauptmann, Ming-Hsuan Yang, Yuan Hao, Irfan Essa, and Lu Jiang. Magvit: Masked generative video transformer. In *CVPR*, 2023. 2

[57] Lijun Yu, José Lezama, Nitesh B. Gundavarapu, Luca Versari, Kihyuk Sohn, David Minnen, Yong Cheng, Vighnesh Birodkar, Agrim Gupta, Xiuye Gu, Alexander G. Hauptmann, Boqing Gong, Ming-Hsuan Yang, Irfan Essa, David A. Ross, and Lu Jiang. Language model beats diffusion–tokenizer is key to visual generation. In *ICLR*, 2024. 6

[58] Qihang Yu, Ju He, Xueqing Deng, Xiaohui Shen, and Liang-Chieh Chen. Randomized autoregressive visual generation. *arXiv preprint arXiv:2411.00776*, 2024. 1, 2, 6

[59] Qihang Yu, Mark Weber, Xueqing Deng, Xiaohui Shen, Daniel Cremers, and Liang-Chieh Chen. An image is worth 32 tokens for reconstruction and generation. *arXiv preprint arXiv:2406.07550*, 2024. 2, 6

[60] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *NeurIPS*, 2019. 5

[61] Qinsheng Zhang, Jiaming Song, Xun Huang, Yongxin Chen, and Ming-Yu Liu. Diffcollage: Parallel generation of large content with diffusion models. In *CVPR*, 2023. 8

[62] Chunting Zhou, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. Transfusion: Predict the next token and diffuse images with one multi-modal model. *arXiv preprint arXiv:2408.11039*, 2024. 1, 2, 4

[63] Xueyan Zou, Jianwei Yang, Hao Zhang, Feng Li, Linjie Li, Jianfeng Wang, Lijuan Wang, Jianfeng Gao, and Yong Jae Lee. Segment everything everywhere all at once. In *NeurIPS*, 2024. 2