Randomized Autoregressive Visual Generation

Supplementary Material

Appendix

The supplementary material includes the following additional information:

- Sec. A provides the detailed hyper-parameters for the final RAR models.
- Sec. B provides the pseudo-code for randomized autoregressive modeling.
- Sec. C provides additional experimental results including RAR's results on the ImageNet-512 benchmark, ablation studies with different visual tokenizers and different training epochs.
- Sec. D visualizes the scan orders used in the ablation study.
- Sec. E provides more visualization samples of RAR models.

A. Hyper-parameters for Final RAR Models

We list the detailed training hyper-parameters and sampling hyper-parameters for all RAR models in Tab. 1.

config	value
training hyper-params	
optimizer	AdamW [4, 7]
learning rate	4e-4
weight decay	0.03
optimizer momentum	(0.9, 0.96)
batch size	2048
learning rate schedule	cosine decay
ending learning rate	1e-5
total epochs	400
warmup epochs	100
annealing start epoch	200
annealing end epoch	300
precision	bfloat16
max grad norm	1.0
dropout rate	0.1
attn dropout rate	0.1
class label dropout rate	0.1
sampling hyper-params	
guidance schedule	pow-cosine [3]
temperature	1.0 (B) / 1.02 (L, XL, XXL)
scale power	2.75 (B) / 2.5 (L) / 1.5 (XL) / 1.2 (XXL)
guidance scale	16.0 (B) / 15.5 (L) / 6.9 (XL) / 8.0 (XXL)

Table 1. **Detailed hyper-parameters for final RAR models.**

B. Pseudo-Code for RAR

We provide a simple pseudo-code of RAR in PyTorch style in Algorithm 1.

Algorithm 1 PyTorch Pseudo-Code for Randomized AutoRegressive (RAR) Modeling

```
class RAR(nn.Module):
  def sample_orders(self, tokens, global_step):
    # sample permutation order at training step global_step.
    orders = []
    # compute the randomized probability r as in Eq. (??).
    prob = 1.0 - min(1.0, max(0.0, (global_step - self.anneal_start) / (self.anneal_end - self.anneal_start)))
    for b in range(tokens.shape[0]):
      if random.random() <prob:
        # random permutation.
        orders.append(torch.randperm(tokens.shape[1]))
      else:
        # raster order (no permutation).
        orders.append(torch.arange(tokens.shape[1]))
    return torch.stack(orders)

  def permute(self, inputs, orders):
    # permute inputs based on orders.
    B, L = inputs.shape[:2]
    indices = torch.arange(B).unsqueeze(1).expand(-1, L)
    return x[indices, orders]

  def forward(self, tokens, condition, global_step):
    # get permutation orders.
    orders = self.sample_orders(global_step, tokens)
    # permute labels for next-token prediction.
    labels = self.permute(tokens.clone(), orders)
    # token embeddings with positional embedding.
    x = self.tok_emb(tokens) + self.pos_emb
    # permute the token orders.
    x = self.permute(x, orders)
    # add target-aware postional embedding as in Eq. (??).
    target_pos_emb = self.target_pos_emb.repeat(x.shape[0], 1, 1)
    target_pos_emb = self.permute(target_pos_emb, orders)
    # shifting so each token will see next-token's embedding.
    target_pos_emb = target_pos_emb[:, 1:]
    x = torch.cat([x[:, :-1] + target_pos_emb, x[:, -1:]], dim=1)
    # transformer forwarding.
    pred = self.transformers(x, condition)
    # next token prediction loss.
    loss = nn.CrossEntropy(pred[:, :-1], labels[:, 1:])
    return loss
```

C. More Experimental Results

We provide additional experimental results on ImageNet-512 in Tab. 2, where RAR demonstrates clear advantages over other methods.

To further ensure a fair comparison excluding the impact of visual tokenizer, we also report RAR's results with the LlamaGen-VQ tokenizer [10]. The comparison against LlamaGen (with the same tokenizer) is provided in Tab. 3. Notably, we did not tune hyper-parameters for RAR-L with LlamaGen-VQ tokenizer, which may not be optimal given the significant differences in vocabulary size (1024

method	#params	FID↓	IS↑
VQGAN [2]	227M	26.52	66.8
MaskGiT [1]	227M	7.32	156.0
DiT-XL/2 [9]	675M	3.04	240.8
DiMR-XL/3R [6]	525M	2.89	289.8
VAR-d36 [11]	2.3B	2.63	303.2
REPA [12]	675M	2.08	274.6
RAR-L	461M	2.35	286.0
RAR-XL	955M	1.83	302.0
RAR-XXL	1.5B	1.66	295.7

Table 2. **RAR on ImageNet-512.** All RAR models were trained for only 200 epochs due to time/resource constraints.

method	#params	FID↓	IS↑
LlamaGen-L-384	343M	3.07	256.1
LlamaGen-XL-384	775M	2.62	244.1
LlamaGen-XXL-384	1.4B	2.34	253.9
LlamaGen-3B-384	3.1B	2.18	263.3
RAR-L	493M	1.95	291.2

Table 3. **RAR *vs.* LlamaGen using the same LlamaGen-VQ tokenizer [10].** Note that LlamaGen generates images at 384 resolutions and resizes back to 256, performing better than directly generating 256 resolutions, which is used by RAR, but with more training/sampling costs.

method	#params	FID↓	epochs
LlamaGen-3B-384 [10]	3.1B	2.18	300
Open-MAGVIT2-XL [8]	1.5B	2.33	350
VAR-d30 [11]	2.0B	1.92	350
MAR-L [5]	479M	1.78	800
RAR-L	461M	1.93	200
RAR-L	461M	1.78	300
RAR-L	461M	1.70	400

Table 4. **FID w/ training epochs.** RAR can perform better against other methods even under a shorter training schedule.

for MaskGIT-VQ and LlamaGen-VQ). However, RAR still outperforms LlamaGen significantly using the same visual tokenizer.

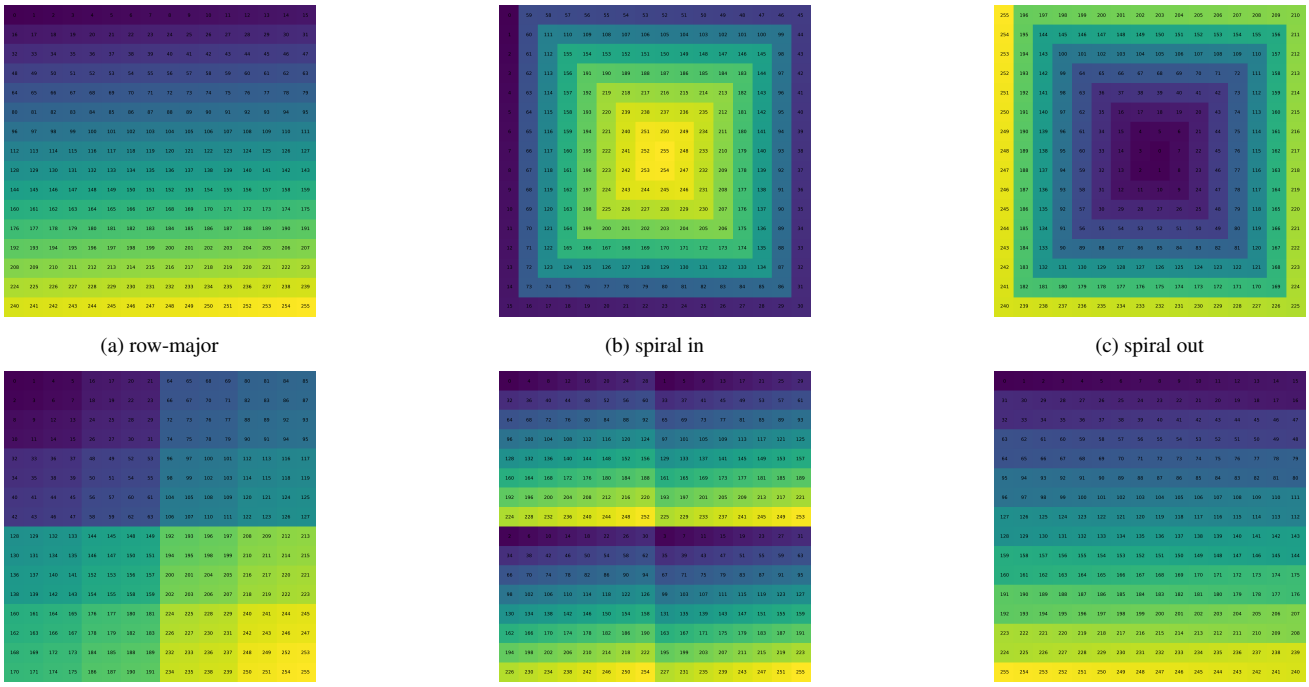
Moreover, we provide a detailed comparison against different methods, along with different training epochs. The results are summarized in Tab. 4. Notably, RAR achieves the best performance among all methods at 400 training epochs, which is comparable to other methods ranging from 300 to 800. In addition, even with a shorter training schedule (*e.g.*, 200 or 300 epochs), we see that RAR still shows clear advantages over other methods. indicating the effectiveness of RAR models.

D. Visualization of Scan Orders

We visualize the 6 scan orders studied in the main paper in Fig. 1.

E. Visualization on Generated Samples

We provide visualization results in Fig. 2, Fig. 3, and Fig. 4.



(a) row-major (b) spiral in (c) spiral out

(d) z-curve (e) subsample (f) alternate

Figure 1. **Different scan orders for a** 16×16 **grid (256 tokens).** The number indicates the token's indices in the scanning order.

Figure 2. **Visualization samples from RAR.** RAR is capable of generating high-fidelity image samples with great diversity.

Figure 3. **Visualization samples from RAR.** RAR is capable of generating high-fidelity image samples with great diversity.

Figure 4. **Visualization samples from RAR.** RAR is capable of generating high-fidelity image samples with great diversity.

References

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

[2] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021. 2

[3] Shanghua Gao, Pan Zhou, Ming-Ming Cheng, and Shuicheng Yan. Masked diffusion transformer is a strong image synthesizer. In *ICCV*, 2023. 1

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 1

[5] Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. *NeurIPS*, 2024. 2

[6] Qihao Liu, Zhanpeng Zeng, Ju He, Qihang Yu, Xiaohui Shen, and Liang-Chieh Chen. Alleviating distortion in image generation via multi-resolution diffusion models. *NeurIPS*, 2024. 2

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *ICLR*, 2019. 1

[8] 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

[9] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023. 2

[10] 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

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

[12] Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think. *ICLR*, 2025. 2