MMAR: Towards Lossless Multi-Modal Auto-Regressive Probabilistic Modeling

Supplementary Material

6. Appendix

6.1. Additional Implementation Details

Table 5. Hyper-parameter settings for MMAR models

Module	Param.	MMAR-0.5B settings	MMAR-7B settings
Diffusion MLP	N_{diff}	8	12
	d_{mlp}	1024	2048
	r_{mlp}	4	4
Image Tonkenizer	Res.	256	256
	d	16	16
EmbeddingViT	N_{ViT}	16	16
	d_{ViT}	1024	1024
LLM's PLoRA	r	512	1280
	α	128	128

Diffusion MLP. Inspired by MAR, we employ a simple MLP architecture to predict $v^{(t)}$, whose detailed architecture is shown in Fig. 6. It consists of a main network and its input/output linear projection layers. The input projection converts the d-dimensional noisy latent $x^{(t)}$ into the mlp hidden dimension d_{mlp}, while the output projection converts the d_{mlp}-dimensional main network output back to d dimensions. The main network is a stack of N_{diff} residual blocks, each comprising an AdaLN [42], followed by a two-layer MLP activated by SiLU. The expand ratio of each MLP is denoted by r_{mlp}, which means its first linear layer projects from d_{mlp} to the dimension of $d_{mlp} \times r_{mlp}$. The condition vector z is added to the diffusion time embedding and is incorporated through AdaLN. In practice, r_{mlp} is set to 4. For MMAR-0.5B, $N_{diff} = 8$ and $d_{mlp} = 1024$, while for MMAR-7B, $N_{diff} = 12$ and $d_{mlp} = 2048$.

Image Tokenizer. MMAR employs the publicly available KL-16 image tokenizer from LDM [47]. This tokenizer processes an 256×256 image into 256 image tokens, each with $d = 16$ channels, and it is kept frozen during training.

EmbeddingViT. The EmbeddingViT module is implemented with a ViT[11] encoder with $N_{ViT} = 16$ layers and $d_{ViT} = 1024$ hidden state channels, processing image tokens into visual embeddings with stronger context awareness. We integrate a learnable position embedding for each image token in EmbeddingViT, corresponding to its 2D position on the image.

LLM. We initialize our LLM using parameters from the open-source QWen2 series models [63]. To preserve text modeling capabilities, we keep the LLM parameters fixed during training and add PLoRA [10] as the image expert, where only the image tokens in the input pass through the introduced LoRA [22] adapters. The PLoRA is applied for each linear layer within the original LLM. Considering the time cost, our ablation study employs QWen2-0.5B-Instruct, with $r = 512$ and $\alpha = 128$ for PLoRA. Furthermore, we use QWen2-7B-Instruct to explore our method's scale-up capability with $r = 1280$ and $\alpha = 128$ for PLoRA. The corresponding MMAR models are denoted by MMAR-0.5B and MMAR-7B, respectively.

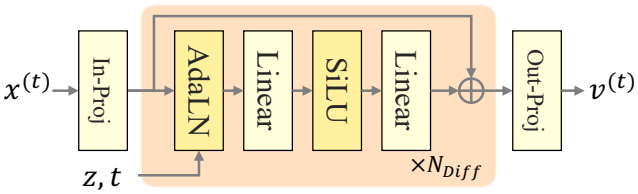


Figure 6. Details of Diffusion MLP architecture.

Projector. We use two MLP-based projectors [67–69] to connect different modules' input and output. Specifically, the PostProjector connects the Image Tokenizer output to the EmbeddingViT's input with a 2-layer MLP, whereas the VisProjector connects EmbeddingViT output to the LLM's input with a 3-layer MLP.

Training Task Proportion. MMAR models $P(I|T)$, $P(T|I)$ and $P(I)$ through the conditional image generation task, the image understanding task and the unconditional image generation task, respectively. Through the first and the second stage training, we set the proportion of these three tasks as 50%, 45%, and 5%, respectively. During the optional third training stage, only image generation tasks are applied, and the proportion of the conditional and unconditional image generation tasks are set to 90% and 10%, respectively.

Mixed-Precision Inference. To enhance numerical stability during image generation, particularly when using small step sizes in the DDPM sampling process, we perform the model's forward pass in `bfloat16` (matching the training precision) but cast the output to `float32` before DDPM sampling. This mitigates potential rounding error without significant computational overhead, improving sampling accuracy efficiently.

Ablation-VQ. Based on the MMAR-0.5B framework, we replace the Image Tokenizer from LDM-KL-16 to LDM-

VQ-16. The image codes extracted using LDM-VQ-16 are then passed through a projector to increase the channel size to match the LLM's hidden size. Subsequently, we add a decoding Linear layer, which takes the hidden states of the LLM's output image portion as input and maps them to the image codebook. The Cross Entropy loss is then calculated between these mapped values and the ground-truth VQ codes.

Transfusion Reimplementation. Our re-implementation version of Transfusion shares most of the MMAR-0.5B's model architecture except for the processing of the model input and output. Following the Transfusion paper, we adopt a input linear projection to convert the output of LDM-KL-16 image tokenizer into the LLM input representation, and use an output linear projection to convert the LLM output into the predicted noise. Noise of different levels are added to the image token input according to different pretraining tasks. For the image-to-text task, the diffusion time step is uniformly sampled within $t \in [0, 500]$, while for the text-to-image task, the diffusion time step is uniformly sampled within $t \in [0, 1000]$. A learnable time embedding corresponding to the time step t is added after the input linear projection. MSE loss is calculated between the predicted and ground-truth noise. During inference, we treat the LLM as a diffusion model, with the condition being the concatenation of the text and the noisy image tokens of the previous diffusion time step $t + 1$.

6.2. Minimizing the Numerical Error in Diffusion Models

To make our discussion clearer, we switch the diffusion noise schedule into an angular form as follows:

$$\begin{cases} \sin\phi_t = \sqrt{1-\bar{\alpha}_t}, \\ \cos\phi_t = \sqrt{\bar{\alpha}_t}. \end{cases} \tag{9}$$

In this way, the forward diffusion process can be written as follows:

$$x^{(t)} = \sqrt{\bar{\alpha}_t}x + \sqrt{1-\bar{\alpha}_t}\epsilon = \cos\phi_t x + \sin\phi_t \epsilon, \tag{10}$$

where $x^{(t)}$, x and ϵ are noised image latent, original image latent and gaussian noise, respectively.

Our goal is to minimize the numerical error term in the DDIM sampling process. However, the form of DDIM sampling process is different under different parameterization method of the diffusion model. Therefore, we need to first define a general form to represent the diffusion model parameterization.

We consider the diffusion model output $u_\theta^{(t)}$ predict a linear combination of data x and noise ϵ, i.e. $u^{(t)} = a_t x + b_t \epsilon$. Note that the coefficients can vary according to the diffusion time step t. Further re-writing the coefficients in the angular form gives:

$$u^{(t)} = r_t \cos\psi_t x + r_t \sin\psi_t \epsilon, \tag{11}$$

where $r_t = \sqrt{a_t^2 + b_t^2}$ represents the scale of $u^{(t)}$. $\cos\psi_t$ and $\sin\psi_t$ balance the proportion of x and ϵ. Combining Eq.10 and Eq.11, we can in turn represent x and ϵ with $u^{(t)}$ and $x^{(t)}$:

$$\begin{cases} x = \frac{\sin\psi_t x^{(t)} - \sin\phi_t u^{(t)}/r_t}{\cos\phi_t \sin\psi_t - \cos\psi_t \sin\phi_t} = \frac{\sin\psi_t x^{(t)} - \sin\phi_t u^{(t)}/r_t}{\sin(\psi_t - \phi_t)}, \\ \epsilon = \frac{\cos\psi_t x^{(t)} - \cos\phi_t u^{(t)}/r_t}{\sin\phi_t \cos\psi_t - \sin\psi_t \cos\phi_t} = -\frac{\cos\psi_t x^{(t)} - \cos\phi_t u^{(t)}/r_t}{\sin(\psi_t - \phi_t)}. \end{cases} \tag{12}$$

Now we consider the general form of DDIM sampling step [51]:

$$x^{(t-1)} = \cos\phi_{t-1}\hat{x}_\theta(x^{(t)}) + \sin\phi_{t-1}\hat{\epsilon}_\theta(x^{(t)}), \tag{13}$$

where $\hat{x}_\theta(x^{(t)})$ and $\hat{\epsilon}_\theta(x^{(t)})$ are the estimated image latent and noise, respectively.

Note that by using Eq.12, both of $\hat{x}_\theta(x^{(t)})$ and $\hat{\epsilon}_\theta(x^{(t)})$ can be derived from the noisy image latent $x^{(t)}$ and the diffusion model output $u_\theta^{(t)}$. Therefore, we can further represent $x^{(t-1)}$ in the following form:

$$\begin{aligned} x^{(t-1)} =& \cos\phi_{t-1}\frac{\sin\psi_t x^{(t)} - \sin\phi_t u_\theta^{(t)}/r_t}{\sin(\psi_t - \phi_t)} \\ &- \sin\phi_{t-1}\frac{\cos\psi_t x^{(t)} - \cos\phi_t u_\theta^{(t)}/r_t}{\sin(\psi_t - \phi_t)} \\ =& \frac{\sin(\phi_{t-1} - \phi_t)u_\theta^{(t)}/r_t - \sin(\phi_{t-1} - \psi_t)x^{(t)}}{\sin(\psi_t - \phi_t)}. \end{aligned} \tag{14}$$

Eq.14 represents the general form of DDIM sampling step under any kind of diffusion model parameterization in the form of Eq.11. To help understanding, we further present the geometric meaning of Eq.14. As shown in Fig.7, term $x^{(t-1)}, x^{(t)}$, and $u_\theta^{(t)}/r_t$ all locate on the unit circle in the $x - \epsilon$ plain. We find that Eq.14 can be interpreted as projecting $x^{(t-1)}$ onto the $(x^{(t)}, \frac{u_\theta^{(t)}}{r_t})$ coordinate system. We illustrate this projection by adding auxiliary line AB and AC. By solving the sine law of $\triangle OBA$ given $OA = 1$, we get:

$$\begin{cases} OB = \frac{\sin(\Delta\phi)}{\sin(\psi_t - \phi_t)} \\ BA = \frac{-\sin(\phi_{t-1} - \psi_t)}{\sin(\psi_t - \phi_t)} \end{cases} \tag{15}$$

By representing $x^{(t-1)} = OB \cdot u_\theta^{(t)}/r_t + AB \cdot x^{(t)}$, we get:

$$x^{(t-1)} = \frac{\sin(\Delta\phi)}{\sin(\psi_t - \phi_t)}u_\theta^{(t)}/r_t - \frac{\sin(\phi_{t-1} - \psi_t)}{\sin(\psi_t - \phi_t)}x^{(t)}, \tag{16}$$

which aligns with Eq.14 given that $\Delta\phi = \phi_{t-1} - \phi_t$.

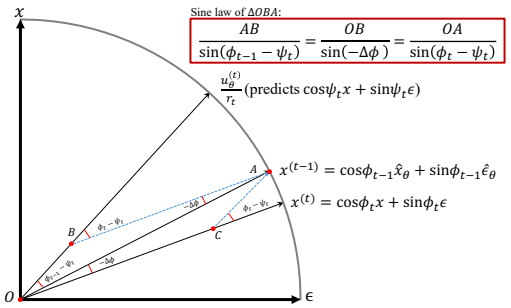


Figure 7. Geometric interpretation of a DDIM sampling step under arbitrary diffusion model parameterization.

Now, we take the numerical error into consideration by multiplying the model output by a factor $1+\delta$, where δ represents the relative error:

$$\tilde{x}^{(t-1)} = \frac{\sin(\phi_{t-1}-\phi_t)(1+\delta)u_\theta^{(t)}/r_t - \sin(\phi_{t-1}-\psi_t)x^{(t)}}{\sin(\psi_t-\phi_t)}. \tag{17}$$

Further, we can isolate the numerical error term from the ideal DDIM sampling step:

$$\tilde{x}^{(t-1)} = x^{(t-1)} + \sin(\phi_{t-1}-\phi_t)\frac{u_\theta^{(t)}/r_t}{\sin(\psi_t-\phi_t)}\delta. \tag{18}$$

From Eq.18, we conclude that the numerical error of a DDIM sampling step is determined by four factors, namely, the step size $\Delta\phi = \phi_{t-1}-\phi_t$, the normalized model output $u_\theta^{(t)}/r_t$, the relative error of the data type δ, and $\sin(\psi_t-\phi_t)$, which is decided by the parameterization of the diffusion model.

Notably, not all these four factors are useful to achieve the goal of minimizing the numerical error. For example, tuning down the step size only decreases the numerical error of each step. As a result, the total step number of DDIM sampling is increased proportionally, which cancels out the effect of error reduction of each single step. The factor $u_\theta^{(t)}/r_t$, which represents the normalized model prediction, is largely decided by the optimization progress and the target data distribution. An approximation can be derived from a perfect prediction, that is $u_\theta^{(t)}/r_t = u^{(t)}/r_t$. In this way,

$$\begin{aligned}\mathbb{E}[(u_\theta^{(t)}/r_t)^2] &\approx \mathbb{E}[(\cos\psi_t x + \sin\psi_t\epsilon)^2] \\ &= \cos^2\psi_t\mathbb{E}[x^2] + \sin^2\psi_t.\end{aligned} \tag{19}$$

In common practice, image tokens x are normalized into unit standard deviation, leading to $\mathbb{E}[(u_\theta^{(t)}/r_t)^2] \approx \cos^2\psi_t + \sin^2\psi_t = 1$. This is a constant number, meaning that there is not much potential to reduce the total numerical error via tuning down $u_\theta^{(t)}/r_t$.

If we decide to scale up our model, it is better to leverage the pre-trained LLMs as well as the highly efficient training infrastructure that is specifically optimized for LLMs. This makes `bfloat16` almost the only choice. As a result, the relative error δ is fixed to $1/128$.

Now, our only choice is to adjust the diffusion model parameterization method, so that $|\sin(\psi_t-\phi_t)|$ is maximized. A simple solution is to set $\psi_t - \phi_t = \pi/2$, resulting in the following parameterization:

$$\begin{aligned}u^{(t)} &= r_t\cos(\phi_t+\pi/2)x + r_t\sin(\phi_t+\pi/2)\epsilon \\ &= r_t(\cos\phi_t\epsilon - \sin\phi_t x).\end{aligned} \tag{20}$$

Note that r_t is still undetermined, which reflects the scale of $u^{(t)}$. From the analysis above, r_t does not affect the numerical error term, since it is canceled out by the normalization of the model output, as seen in the factor $u_\theta^{(t)}/r_t$. Therefore, r_t can be chosen freely, or based on other considerations. We consider that the smooth optimization of a neural network often requires the activation and output not too large or small. Therefore, we require a unit standard deviation for $u^{(t)}$, making $r_t = 1$ constantly.

The final parameterization of our diffusion model is as follows:

$$u^{(t)} = \cos\phi_t\epsilon - \sin\phi_t x. \tag{21}$$

We notice that this parameterization is coincidentally the "v-prediction" parameterization [49]. Note that, however, "v-prediction" is initially proposed for the efficient distillation of diffusion models, rather than reducing the numerical error of diffusion models. To the best of our knowledge, our work is the first to derive "v-prediction" parameterization from the first principle of minimizing the numerical error in diffusion models.

6.3. Deriving Theoretical Numerical Error for ϵ-Prediction Models

The ϵ-prediction parameterization corresponds to $\psi_t = \frac{\pi}{2}$ in the angular parameterization form given by Eq.11. Substituting $\psi_t = \frac{\pi}{2}$ and $u_\theta^{(t)}/r_t = \epsilon_\theta$ into Eq.18, we get:

$$\tilde{x}^{(t-1)} = x^{(t-1)} + \sin(\phi_{t-1}-\phi_t)\frac{\epsilon_\theta}{\cos(\phi_t)}\delta. \tag{22}$$

Further, we cancel out the step size factor $\sin(\phi_{t-1}-\phi_t)$ within the numerical error term, only focusing on "the numerical error introduced per **unit** DDIM step":

$$e^{(t)} = \frac{\epsilon_\theta}{\cos\phi_t}\delta. \tag{23}$$

Next, we will show that $e^{(t)}$ can also be interpreted as the equivalent v-prediction numerical error for an ϵ-prediction model.

For an ϵ-prediction model, $u_\theta^{(t)} = \epsilon_\theta$. In order to calculate the equivalent $v_\theta^{(t)}$ value, we need to represent $v_\theta^{(t)}$ with

Table 6. Detailed visual understanding evaluation results.

Benchmark	Chameleon-7B	Transfusion*	Show-o	MMAR-0.5B	MMAR-0.5B w/ ϵ-pred.	MMAR-0.5B w/ VQ	MMAR-7B	MMAR-7B w/ Stg3
AI2D [24]	34.81	40.22	32.48	43.43	41.90	41.90	**64.64**	63.54
ChartQA [40]	3.84	9.56	11.32	10.20	10.36	9.36	**13.64**	12.52
DocVQA [41]	1.51	6.72	**18.24**	7.62	6.77	6.79	11.12	10.62
Hallu.Bench [18]	39.01	41.54	40.90	42.80	41.11	41.54	**53.10**	52.05
MathVista [39]	21.90	22.60	23.20	21.60	23.10	22.90	**32.40**	31.30
MMBench^{CN} [35]	10.14	27.23	0.52	43.99	38.83	31.87	**70.53**	66.75
MMBench^{EN} [35]	13.32	29.47	42.44	48.45	45.53	37.54	**70.45**	67.78
MME^{P} [15]	125.8	594.3	1182.7	882.1	880.7	618.2	**1486.9**	1421.9
MME^{C} [15]	33.9	206.1	225.0	256.8	232.1	273.2	268.9	**303.6**
MMMU [66]	24.00	29.33	26.44	29.33	25.33	29.67	41.33	**47.33**
MMStar [5]	20.47	28.13	32.00	32.13	31.07	28.07	**41.87**	40.87
MMVet [65]	7.34	13.90	20.87	18.49	17.98	14.45	**30.64**	29.17
OCRBench [36]	0.50	2.30	15.20	18.70	7.10	2.10	**25.80**	21.20
POPE [31]	30.86	66.90	**84.50**	70.74	71.14	66.98	84.02	83.21
RealWorldQA	27.06	36.99	27.97	38.30	35.16	36.60	**53.59**	52.68
ScienceQA [38]	44.83	45.92	41.82	47.54	45.21	45.35	**74.39**	73.20
SEEDBench [26]	34.61	42.40	51.61	55.70	53.72	44.93	**68.63**	66.59
TextVQA [50]	5.43	9.94	**38.35**	16.77	12.40	9.46	24.37	21.35
Average	18.34	28.26	33.06	34.56	32.21	29.70	**48.25**	47.18

the predicted ϵ_θ and the known $x^{(t)}$, which is calculated as follows:

$$\begin{aligned} v_\theta^{(t)} &= \cos\phi_t\epsilon_\theta - \sin\phi_t\hat{x}_\theta(x^{(t)}) \\ &= \cos\phi_t\epsilon_\theta - \sin\phi_t\frac{x^{(t)} - \sin\phi_t\epsilon_\theta}{\cos\phi_t} \\ &= \frac{\epsilon_\theta}{\cos\phi_t} - \tan\phi_t x^{(t)}. \end{aligned} \tag{24}$$

Considering the numerical error, we get:

$$\tilde{v}_\theta^{(t)} = \frac{\epsilon_\theta(1+\delta)}{\cos\phi_t} - \tan\phi_t x^{(t)} = v_\theta^{(t)} + \frac{\epsilon_\theta}{\cos\phi_t}\delta. \tag{25}$$

Note that the numerical error term in the above equation is exactly $e^{(t)}$, proving that $e^{(t)}$ can be interpreted as the equivalent v-prediction numerical error for an ϵ-prediction model.

Taking numerical error effect into the v-prediction-based diffusion loss, we get:

$$\begin{aligned} \mathbb{E}[(v^{(t)} - \tilde{v}_\theta^{(t)})^2] =& \mathbb{E}[(v^{(t)} - v_\theta^{(t)} - e^{(t)})^2] \\ =& \mathbb{E}[(v^{(t)} - v_\theta^{(t)})^2] - 2\mathbb{E}[(v^{(t)} - v_\theta^{(t)})e^{(t)}] \\ &+ \mathbb{E}[(e^{(t)})^2]. \end{aligned} \tag{26}$$

Due to the fact that numerical error $e^{(t)}$ is independent from the training loss and that the expectation of $e^{(t)}$ is 0, we get $\mathbb{E}[(v^{(t)} - v_\theta^{(t)})e^{(t)}] = 0$. Therefore, the only numerical error term is $\mathbb{E}[(e^{(t)})^2]$. Given that the standard deviation of ϵ_θ is 1, and considering that we use `bfloat16` as training data type, which means $\delta = 1/128$, we get

$$\mathbb{E}[(e^{(t)})^2] = 1/(128\cos(\phi_t))^2 = 1/(128^2\bar{\alpha}_t). \tag{27}$$

This is the theoretical numerical error of the v-prediction diffusion loss for an ϵ-prediction model.

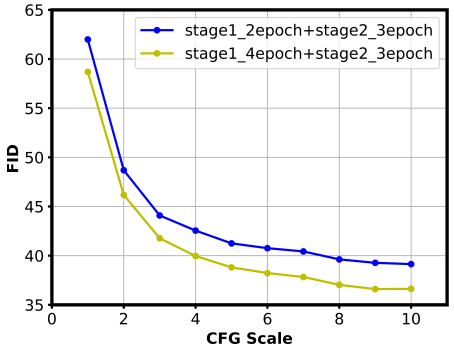


Figure 8. The impact of CFG scale on image generation quality.

6.4. CFG With v-prediction

From Equation $v_i^{(t)} = \sqrt{\bar{\alpha}_t}\epsilon - \sqrt{1-\bar{\alpha}_t}x_i$, we can derive the following equation.

$$\epsilon = \sqrt{1-\bar{\alpha}^{(t)}}x^{(t)} + \sqrt{\bar{\alpha}^{(t)}}v \tag{28}$$

For the CFG of ϵ, it can be simplified as follows.

$$\begin{aligned} \epsilon =& \epsilon_u + \omega(\epsilon_c - \epsilon_u) \\ =& \sqrt{1-\bar{\alpha}^{(t)}}x^{(t)} + \sqrt{\bar{\alpha}^{(t)}}v_u + \omega\sqrt{\bar{\alpha}^{(t)}}(v_c - v_u) \\ =& \sqrt{1-\bar{\alpha}^{(t)}}x^{(t)} + \sqrt{\bar{\alpha}^{(t)}}(v_u + \omega(v_c - v_u)) \end{aligned} \tag{29}$$

Ultimately, we obtain $v = v_u + \omega(v_c - v_u)$. The CFG of v and ϵ are equivalent.

6.5. Impact of CFG Scaling

We select models from the second and fourth epochs of the first stage as starting points for the second stage, train them for 3 epochs, and then test the MSCOCO FID-30K under varying CFG intensities. As shown in Fig. 8, our method achieves better FID scores as the CFG scale increases from 1 to 10. It is worth noting that most probabilistic generative models typically have a CFG scale between 1.5 and 5. Additionally, it is observed that a longer training duration in

Figure 9. Generated images from MMAR-7B after the third training stage (1/2).

the first stage consistently results in better generation outcomes at all CFG scales.

6.6. Detailed Visual Understanding Evaluation Results

A total of 18 visual understanding benchmarks from VLMEvalKit [12] are used to evaluate MMAR models comprehensively. The evaluation is also conducted on the existing joint image-text probabilistic models using the publicly available checkpoints[3][4]. The detailed evaluation results are shown in Table 6. All scores have been scaled to a range of 0 to 100 except that we show the original score of MME benchmarks. The average score is calculated on the normalized score of all the benchmarks including MME.

6.7. Examples:Image Generation Sample

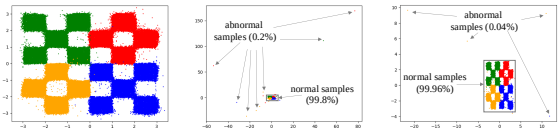
In Fig.9, 10, we showcase some generated examples from MMAR-7B after the third stage training, featuring animals, plants, real-world scenes, artistic scenes, and counterfactual themes.

6.8. Numerical Error Empirical Validation

We provide direct empirical demonstration using a 2D chessboard distribution (class-conditioned quadrant map-

[3]https://huggingface.co/facebook/chameleon-7b
[4]https://huggingface.co/showlab/show-o-w-clip-vit

Figure 10. Generated images from MMAR-7B after the third training stage (2/2).



(a) v-prediction + bf16 (all 1M samples) (b) ϵ-prediction + bf16 (all 10k samples) (c) ϵ-prediction + fp32 (all 10k samples)

Figure 11. Diffusion sampling results with different prediction targets and training precision. Figures are scaled differently to include all generated samples.

ping). As Fig.11 reveals: (1) v-prediction maintains stability even with bf16 precision at scale (1M samples, Fig.(a)), achieving $< 10^{-6}$ token error rate. (2) ϵ-prediction exhibits visible artifacts (Fig.(b),(c)) with 0.2% token error rate (bf16), equivalent to 51.2% image-level failure for 256-token images. (3) Precision elevation (fp32) reduces artifacts to 0.04%. These observations validate v-prediction's critical role in mitigating low-precision training risks.