Denoising, Fast and Slow: Difficulty-Aware Adaptive Sampling for Image Generation

Supplementary Material

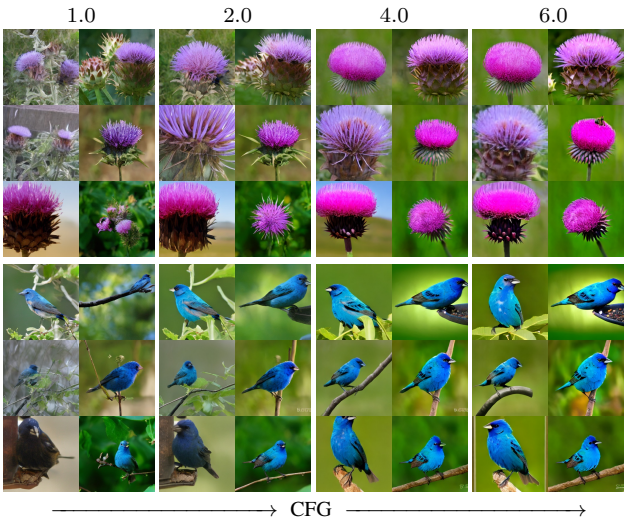

Figure S1. **Uncurated Imagenet** 256×256 **samples** with increasing CFG scale from our REPA-PF-XL model. We use 25 outer steps and 4 inner steps, resulting in a total of 100 NFE with our *dual-loop* sampler.

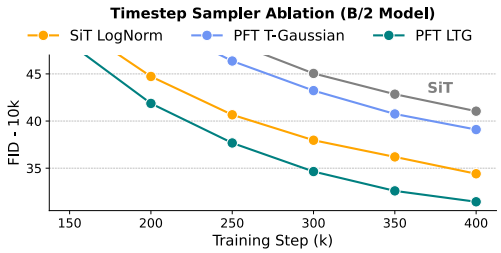

Figure S2. **Ablation of timestep samplers** under a fixed NFE budget of 100. Our proposed PFT with Logit-Normal Truncated Gaussian (LTG) shows orthogonal gains and consistently outperforms the SiT Logit-Normal [16] and the Gaussian samplers.

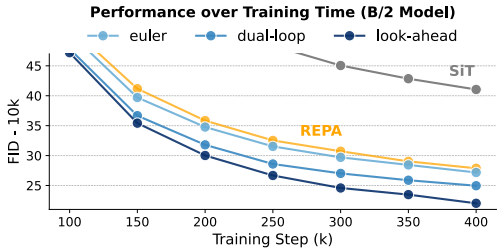

Figure S3. **Representation Alignment** comparison on B/2 models. Patch Forcing (blue lines) is orthogonal to REPA [57] and we can observe similar training improvements compared to standard SiT training [41]. Integrating REPA into our PFT model yields further gains, while our *dual-loop* and *look-ahead* samplers both improve upon baseline Euler sampling with our REPA-PF model. We keep all NFEs fixed to ensure a fair comparison.

A. Further Results

A.1. Class-conditional Generation

Comparison of Timestep Schedulers We compare different timestep schedulers in Fig. S2 to identify the source of the gains from our PFT Logit-Normal Truncated Gaussian (LTG) sampler. To disentangle the individual effects, we compare against the SiT baseline with Logit-Normal timestep sampling, verifying that the improvement is not solely due to the Logit-Normal schedule, and against a Patch Forcing model with a simple truncated Gaussian sampler, verifying that the gain is not merely due to Patch Forcing. PFT with LTG consistently outperforms both baselines, showing that the benefits of Logit-Normal timestep sampling and truncated Gaussian patch-wise timestep allocation are complementary.

Orthogonality to REPA Figure S3 shows performance over training iterations when integrating REPA into PF and compares it to SiT [41] and REPA [57]. Our PFT consistently improves over REPA, yielding additive gains already under standard Euler sampling. Applying our uncertainty-aware samplers on top provides further improvements. For a fair comparison, we keep the number of function evaluations (NFE) fixed across all sampling strategies.

Difficulty Percentile at Sampling We ablate the confidence threshold used to select patches for early progression in the PFT-B/2 model across different sampling strategies in Fig. S4. Specifically, we vary the percentile cutoff that determines which patches are considered "confident" and thus used to provide context for others. Interestingly, random selection can offer slight improvements over the parallel sampling baseline, suggesting that even naive context can help. However, the benefits are substantially greater when confident patches are selected based on predicted uncertainty. Both of our proposed samplers: *dual-loop* and *look-ahead* samplers, consistently outperform random sampling and parallel sampling, confirming the value of informed, uncertainty-driven patch scheduling.

Inner vs. Outer Steps In the dual-loop sampler, we can freely choose the number of inner vs outer steps, as well

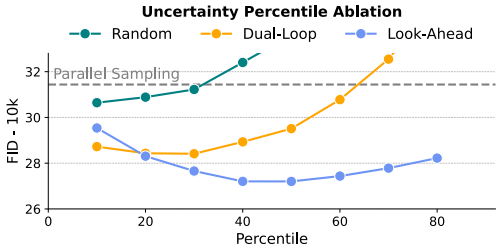

Figure S4. **Ablation of patch-difficulty threshold** with Patch Forcing B/2 model. We ablate over the percentile for confident pixels at sampling. Our samplers perform best at around the 40% percentile and outperform the parallel sampling baseline.

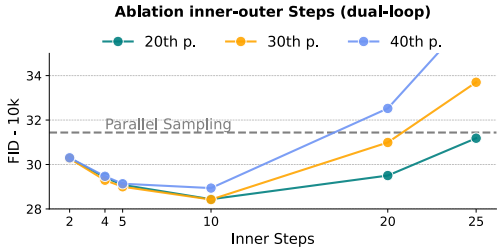

Figure S5. **Ablation of dual-loop hyperparameters** with Patch Forcing B/2 model. We ablate over percentiles of confident patches and also vary the number of inner vs outer steps while keeping the total NFE at inner $\times$ outer $= 100$. The model achieves optimal performance when the inner and outer steps are balanced.

Algorithm S1 Look-ahead Sampling

Require: Model $f_\theta(x,t) \to (v,\ \mathbf{uc})$ where v is velocity and $\mathbf{uc}$ is uncertainty map, $x \in \mathbb{R}^{B\times C\times H\times W}$, $t \in \mathbb{R}^{B\times H\times W}$; uncertainty percentile $p \in (0,1)$; context factor $\alpha > 1$; time grid $0 = t_0 < t_1 < \cdots < t_K = 1$; Euler stepper $\mathrm{Step}(x,t,t') := x + (t'-t)\,v$ (*replace with other ODE solver if desired*)

Initialize $x_0 \sim \mathcal{N}(0, I)$ ▷ start from noise at $t_0 = 0$
for $k = 0$ to $K-1$ **do**
  $x \leftarrow x_k, \quad t \leftarrow t_k, \quad t_{\text{next}} \leftarrow t_{k+1}$
  $(v_t,\ \mathbf{uc}_t) \leftarrow f_\theta(x,t)$ ▷ predict velocity and uncertainty
  $\tau_p \leftarrow \mathrm{Percentile}(u_c,\ p)$ ▷ adaptive thresholding
  $M_{\text{conf}} \leftarrow \mathbf{1}[\,\mathbf{uc} \le \tau_p\,];\ M_{\text{unc}} \leftarrow 1 - M_{\text{conf}}$ ▷ confidence masks
  $t_{\text{ctx}} \leftarrow \min(\alpha\, t,\ 1)$
  $x_{\text{ctx}} \leftarrow \mathrm{Step}(x,\ t,\ t_{\text{ctx}})$ ▷ one-step look-ahead
  $\tilde{x} \leftarrow M_{\text{conf}} \odot x_{\text{ctx}} + M_{\text{unc}} \odot x$ ▷ use look-ahead for M_{conf}
  $\tilde{t} \leftarrow M_{\text{conf}} \odot t_{\text{ctx}} + M_{\text{unc}} \odot t$ ▷ use look-ahead for M_{conf}
  $(v_{\text{ctx}},\ \_\_) \leftarrow f_\theta(\tilde{x},\ \tilde{t})$ ▷ context-aware velocity
  $v_{\text{final}} \leftarrow M_{\text{unc}} \odot v_{\text{ctx}} + M_{\text{conf}} \odot v$ ▷ replace M_{unc} prediction
  $x \leftarrow x + (t_{\text{next}} - t)\, v_{\text{final}}$ ▷ advance to t_{k+1}
end for
return x

as the percentile of confident patches. We conduct ablations under a fixed NFE $= 100$ in Figure S5. We vary the number of inner and outer loop steps to identify the optimal configuration, and ablate over different confidence percentiles used to select the patches for context propagation. We find that using 10 inner steps and 10 outer steps yields the best performance. When the number of outer steps is too high, the behavior begins to resemble standard Euler sampling, diminishing gains from the uncertainty-aware sampling. Conversely, too few outer steps lead to overly aggressive updates, resulting in inaccurate context and degraded performance. This effect is particularly pronounced when a larger percentage (40%) of confident pixels are advanced with insufficient outer steps, as reflected by a sharp drop in generation quality.

Classifier-free Guidance We show that our method also works well with standard classifier-free guidance (CFG) [21]. In Fig. S1 we show, similar to previous findings, increasing the CFG scale leads to visually better results, but at the cost of reduced diversity.

Sampling Ratio (%)	SiT-B 100	+RAS 60	PF-B 100	+Dual-loop 100	+Dual-loop,Cache 60
NFE=16	48.0	50.9	43.9	42.5	45.0
NFE=32	43.1	45.2	33.9	32.4	36.3
NFE=250	41.0	41.2	31.1	28.2	28.5

Table S1. **ImageNet FID$_{10K}$ compute-fidelity tradeoff.** PF + caching outperforms RAS with identical model and compute.

Connection to Region-Adaptive Sampling As discussed in related works, RAS [37] is fundamentally an inference-time acceleration method. It focuses on *computational efficiency during inference via caching* and reusing patch computations across timesteps with *minor fidelity degradation* at fixed NFEs. In contrast, PF introduces a training paradigm: we train the model with patch-wise timesteps, enabling it to reason over heterogeneous noise states, which already improves generation quality. Building on this, we design samplers that improve image fidelity further by propagating confident patches to provide context for harder ones. These components are designed for performance gains rather than acceleration. Thus, our method and RAS are *orthogonal and can be combined*: Tab. S1 shows that our method integrates with previous-step prediction caching and KV caching from RAS to achieve a favorable compute-performance trade-off, even with a simple integration of our dual-loop sampler (last column).

A.2. Text-conditional Generation

Text Rendering Quality. Figure S6 presents uncurated samples from two text-to-image models: one trained with standard Flow Matching (FM) and one trained with Patch Forcing. Following [16], we use logit-normal timestep sampling for the FM baseline, while the Patch Forcing Transformer (PFT) is trained with our proposed Logit-Normal Truncated Gaussian (LTG) sampler. Both models share exactly the same architecture, text encoder stack, and training data; the only difference is the transition from Flow Match-

Model	Exact Match Rate ↑	Mean Lev. ↓	Mean Norm. Lev. ↓
Flow Matching	0.3937	9.5400	0.3348
PFT + Euler	0.6162	5.5462	0.2221
PFT + Dual Loop	0.6125	5.8300	0.2226
PFT + Look Ahead	0.4875	4.7313	0.2911

Table S2. **OCR-based text rendering comparison** for a Flow Matching Baseline with our Patch Forcing Transformer. Both models share the same architecture, text encoder stack, number of parameters, data, and number of function evaluations.

	Color B-VQA	Shape B-VQA	Texture B-VQA	2D-Spatial UniDet	3D-Spatial UniDet	Non-Spatial CLIP
Euler	0.3186	0.3264	0.3394	0.0530	0.1793	0.2793
Dual-loop	**0.3285**	**0.3371**	**0.3448**	0.0608	0.1753	0.2782
Look-ahead	0.3126	0.3313	0.3348	**0.0625**	**0.1925**	**0.2814**

Table S3. **Evaluation of sampling strategies on PixArt-α.** We evaluate sampling strategies **zero-shot** on the pre-trained PixArt-α model [12] using the T2I-CompBench++ benchmark [24], where the model is not trained with varying patch-wise timesteps. All experiments are evaluated without CFG.

ing training to Patch Forcing training with heterogeneous patch-wise noise levels. Overall, we observe that our PFT produces clearer text compared to the FM baseline.

We further evaluate this effect quantitatively by measuring text rendering accuracy using an OCR-based evaluation protocol. Specifically, we construct a set of prompts that explicitly require rendering text (e.g., "A man holding a sign that reads '…'") and generate images from both models. We then apply an off-the-shelf OCR model (EasyOCR) to extract the rendered text and compare it to the ground-truth prompt text. We generate the prompt set based on 10 template texts and 10 inner texts, and sample 8 images per prompt, resulting in 800 evaluation images. Table S2 shows that our PFT consistently improves text rendering quality over the Flow Matching baseline. The qualitative improvements observed in Figure S6 are also reflected quantitatively by higher exact match rates and lower Levenshtein distances. In particular, PFT with Euler and Dual Loop sampling achieves substantial gains across all metrics. Interestingly, the Look-Ahead sampler, while often performing favorably on standard image quality metrics, shows the weakest text rendering performance among the PFT variants. A possible explanation is that, in the Look-Ahead setting, the context is fixed during generation, limiting the model's ability to iteratively refine and correct text. In contrast, Dual Loop sampling allows for limited inner updates, which might explain why its performance remains closer to the Euler baseline.

Adapting a Pre-Trained T2I Model. We additionally explore whether our samplers can be applied *zero-shot* to an existing pretrained text-to-image model. Since our method requires a patch-difficulty prediction, PixArt-α [12] is one of the few suitable candidates, as it inherits a σ-prediction head from the original Diffusion Transformer [44]. Hence, we repurpose the model's variance prediction as a proxy for patch difficulty by averaging it across channels, which we find to align with difficult regions (see Figure S7). Although PixArt-α is not trained with patch-wise timesteps, we find that it is relatively robust to spatially varying noise scales (see Figure S7). Therefore, to apply our sampling strategy, we broadcast the timestep conditioning to the spatial token level such that each latent patch can be assigned its own noise level. Taken together, this enables uncertainty-aware sampling without any weight modification or fine-tuning. Tab. S3 shows that applying our samplers zero-shot to PixArt-α improves image quality over the standard Euler sampler on T2I-CompBench++ [24]. Although this analysis is exploratory, it suggests that diffusion transformers can tolerate spatially varying noise levels even when trained only with homogeneous timesteps. Patch Forcing amplifies this effect by explicitly training with patch-wise noise scales and thereby reducing the train–test gap, which makes our samplers more effective.

Comparison to Self-Flow Concurrent to our work, *Self-Flow* [10] also introduces heterogeneous noise levels during diffusion training. Similar to our findings, the authors observe that naively applying diffusion forcing at the patch/-token level substantially degrades generation quality. To address this issue, Self-Flow proposes Dual-Timestep sampling, which uses only two distinct noise scales during training. While this reduces the train-test gap, we approach the issue from a different perspective. Sampling the timesteps per patch from a uniform distribution results in an average noise level around $t \approx 0.5$ during training, whereas at inference, the model must start from full noise, with no prior information available. To close this gap, we directly control the maximum information during training via our LTG sampler. Furthermore, while Self-Flow primarily leverages heterogeneous noise levels for representation learning, we additionally show that they can be exploited for non-uniform denoising through our proposed sampling strategies.

Similar to Self-Flow, we also observe improved text rendering for our T2I model trained with Patch Forcing (Section A.2). This suggests that the gains may stem from heterogeneous noise scales during training rather than the specific representation loss used in Self-Flow, which may render the additional teacher–student forward passes unnecessary. Understanding what exactly drives these improvements is an interesting direction for future work.

B. Implementation Details

We detail all implementation details in the following sections for ImageNet training and our text-to-image experi-

Figure S6. **Uncurated Text-to-Image** 256 **px samples** from the baseline Flow Matching model with Logit-Normal schedule compared to our PFT model. We keep the model architecture, data, training, and sampling fixed (same amount of Euler steps and seed) and only change the training paradigm from plain Flow Matching to Patch Forcing.

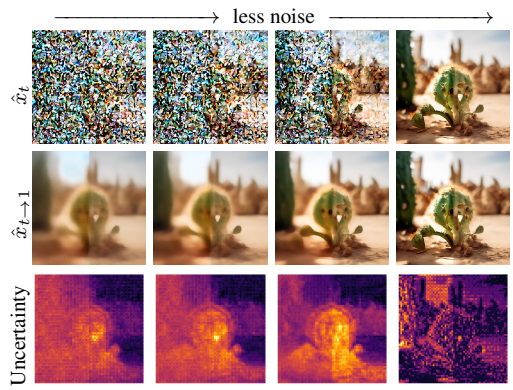

Figure S7. **Repurposing PixArt-α for heterogeneous timesteps.** We adapt a pretrained text-to-image model (PixArt-α [12]) and observe that it can handle per-patch noise levels to some degree. The σ prediction of it can be repurposed for our samplers. We do not use CFG in this example.

ments.

B.1. Class-conditional ImageNet

We follow the training setup of [41], using a batch size of 256 and AdamW [38] with a constant learning rate of 1×10^{-4}. We maintain an exponential moving average (EMA) of model parameters throughout training with a decay of 0.9999, and report results using the EMA weights. The only data augmentation applied is random horizontal flipping with probability 0.5. For metrics, we follow the ADM evaluation suite [13]. Results reported in Tab. 2 are from our PF-XL model with representation alignment [57].

The original SiT implementation [41] requires only minimal changes to support patch-wise noise levels. In Fig. S12 we outline the necessary modifications to the Diffusion Transformer. With these adjustments in place, the only remaining change is to adapt the timestep sampler used during training.

LTG Scheduler Implementation Details As discussed in the paper, we design a truncated Gaussian sampler that samples only from the lower (noisier) half of the Gaussian distribution. However, since timesteps must remain within the valid range $[0, 1]$, we cannot directly apply arbitrary combinations of $\mathrm{t}_{\max}$ and standard deviation std without risk of sampling invalid values. To ensure the sampled timesteps remain within a meaningful range, we dynamically adjust the standard deviation based on $\mathrm{t}_{\max}$. Specifically, we set $\mathrm{std}_{\text{eff}} = \min(\mathrm{t}_{\max}/2, \mathrm{std})$, so that, according to the empirical 2-standard-deviation rule, approximately 95% of the mass of the Gaussian distribution lies above 0. For rare cases where sampled values fall below 0, we replace them with random values uniformly sampled from $[0, t_{\max}]$.

Algorithm S2 LTG Sampler Pseudo-code

```
# loc and scale: logit-normal params for t_max
# std: Gaussian spread for t

t_max = lognorm(loc + scale * randn(bs))
std = min(t_max / 2, std)

t_max = t_max[:, None]
std = std[:, None]
eps = randn(bs, dim)

t = t_max - abs(eps) * std
t[t < 0] = rand_like(t) * t_max # reset negatives

return t
```

For our LTG sampler we have three parameters. First, the location m and scale s parameters for the Logit-Normal $t_{\max}$ sampling (see Figure S8 left) and second, the σ parameter controlling the spread of the lower half around the sampled $t_{\max}$. We provide the pseudo-code in Algorithm S2.

B.2. Text-to-Image

For our text-to-image experiments, we train a 1.2B Patch Forcing Transformer (PFT) on a 120M subset of COYO [7]. We first recaption all images with InternVL3-2B [62] using long-form descriptions, and then distill this long caption into three variants: *long*, *medium*, and *keyword* captions. During training, we sample uniformly from these caption variants, and with probability 0.1, we replace the caption with an empty prompt to enable classifier-free guidance [21]. We encode text with Qwen3-1.7B [32], and following [40], we insert a lightweight two-layer text refiner transformer of width 1536 between the frozen text features and the cross-attention blocks of our diffusion transformer. For all T2I models, we use the FLUX.2 autoencoder [30].

Similar to prior work, such as SDXL [45], we employ crop-size conditioning during training. Since our PFT uses RoPE [52], we directly integrate crop-size conditioning via positional encoding, adapting relative positions to the sampled crop. At inference time, we always set the crop size to the full image resolution. Figure S10 shows how this conditioning mechanism can be used to generate different crop-outs for the same prompt.

We pre-train PFT for 400k iterations at 256 px resolution with a batch size of 1024 and a fixed learning rate of 10^{-4}, and then finetune it for an additional 50k iterations at 512 px resolution on a mixture of high-aesthetics filtered COYO and JourneyDB [53]. For all models, we maintain an EMA with a decay factor of 0.9999 and report results using the EMA weights. We set the uncertainty loss weight to 0.01 for all experiments, following [56].

For the comparison in Figures S6 and 13, we additionally train a vanilla Flow Matching baseline on exactly the same data, with the same architecture, batch size, learning rate, and training schedule. The only difference lies in the

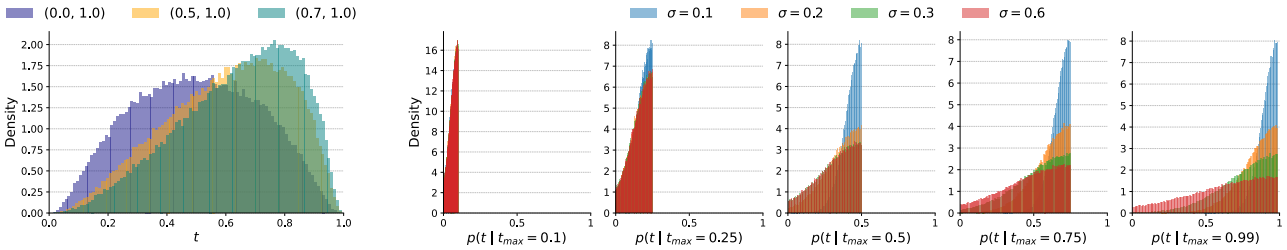

Figure S8. **Logit-Normal Truncated Gaussian Timestep Schedule** *Left*: We first sample t_{max} from a Logit-Normal distribution with location and scale parameters (m, s) according to [16]. *Right*: Given t_{max}, we sample the individual patch timesteps according to a truncated Gaussian with parameter σ.

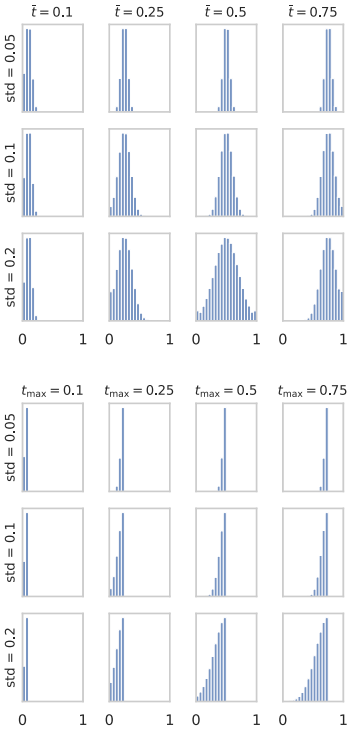

Figure S9. **Visualization of our timesteps samplers**: the first is a Gaussian distribution-based sampler, while the second is a truncated Gaussian sampler.

training timestep sampling strategy: the vanilla model uses the logit-normal schedule from [16] and broadcasts a single timestep to all spatial tokens, whereas the PFT samples different timesteps per patch using our Logit-Normal Truncated Gaussian sampler.

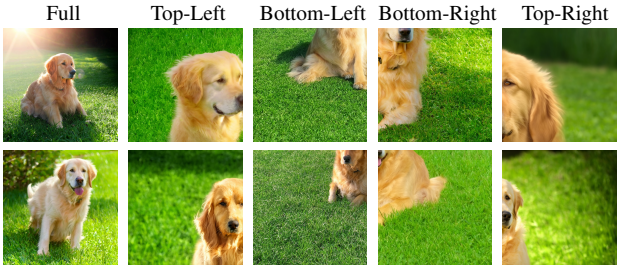

Figure S10. **Crop-size conditioning** via RoPE.

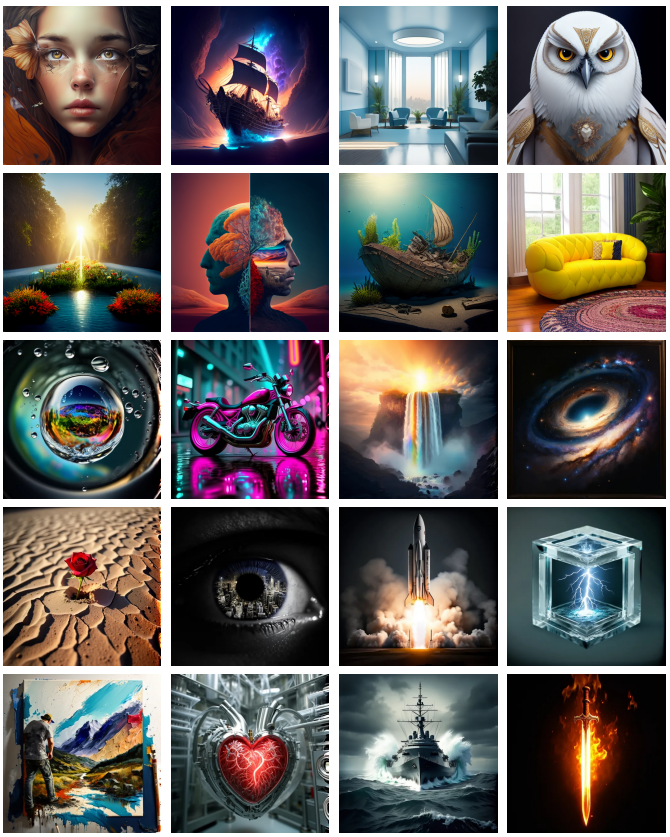

Figure S11. **Qualitative text-to-image results.**

```
 # Original file: https://github.com/willisma/SiT/blob/main/models.py

 def modulate(x, shift, scale):
-    return x * (1 + scale.unsqueeze(1)) + shift.unsqueeze(1)
+    return x * (1 + scale) + shift

 ...

 class SiTBlock(nn.Module):
     ...

     def forward(self, x, c):
-        shift_msa, scale_msa, gate_msa, shift_mlp, scale_mlp, gate_mlp = self.adaLN_modulation(c).chunk(6, dim=1)
-        x = x + gate_msa.unsqueeze(1) * self.attn(modulate(self.norm1(x), shift_msa, scale_msa))
-        x = x + gate_mlp.unsqueeze(1) * self.mlp(modulate(self.norm2(x), shift_mlp, scale_mlp))
+        shift_msa, scale_msa, gate_msa, shift_mlp, scale_mlp, gate_mlp = self.adaLN_modulation(c).chunk(6, dim=-1)
+        x = x + gate_msa * self.attn(modulate(self.norm1(x), shift_msa, scale_msa))
+        x = x + gate_mlp * self.mlp(modulate(self.norm2(x), shift_mlp, scale_mlp))
         return x
 ...

 class FinalLayer(nn.Module):
     ...

     def forward(self, x, c):
-        shift, scale = self.adaLN_modulation(c).chunk(2, dim=1)
+        shift, scale = self.adaLN_modulation(c).chunk(2, dim=-1)
         x = modulate(self.norm_final(x), shift, scale)
         x = self.linear(x)
         return x

 ...

 class SiT(nn.Module):
     def __init__(self, ...):
         ...
-        self.out_channels = in_channels
+        self.out_channels = in_channels + 1

     def forward(self, x, t, y):
         """
         t: (b, n) with n = number of tokens
         """
         x = self.x_embedder(x) + self.pos_embed

-        t = self.t_embedder(t)
+        # patch-level t's
+        t = t[..., None]                        # (b, n) -> (b, n, 1)
+        t = self.t_embedder(t)                  # (b, 1, n, d)
+        t = t.squeeze(1)                        # (b, n, d): one embedding per patch

-        y = self.y_embedder(y, self.training)
+        # broadcast y-embedding
+        y = self.y_embedder(y, self.training)   # (b, c)
+        y = y.unsqueeze(1)                      # (b, 1, c)

         c = t + y
         for block in self.blocks:
             x = block(x, c)
         x = self.final_layer(x, c)
         x = self.unpatchify(x)

-        return x
+        # split uncertainty
+        logvar_theta = x[:, -1:, :, :]          # (b, 1, h, w)
+        x = x[:, :-1, :, :]                     # (b, c, h, w)
+        return x, logvar_theta
```

Figure S12. **Code changes** to adapt the original PyTorch SiT/DiT architecture [41, 44] to allow different noise scales per patch. Original file sourced from https://github.com/willisma/SiT/blob/main/models.py.

29: Axolotl

207: Golden Retriever

94: Hummingbird

295: American Black Bear

130: Flamingo

299: Meerkat

238: Greater Swiss Mountain Dog

309: Bee

437: Beacon

327: Starfish

Figure S13. **Uncurated Imagenet** 256×256 **samples** from our REPA-PF-XL model. We use 100 NFE and a CFG value of 2.5.