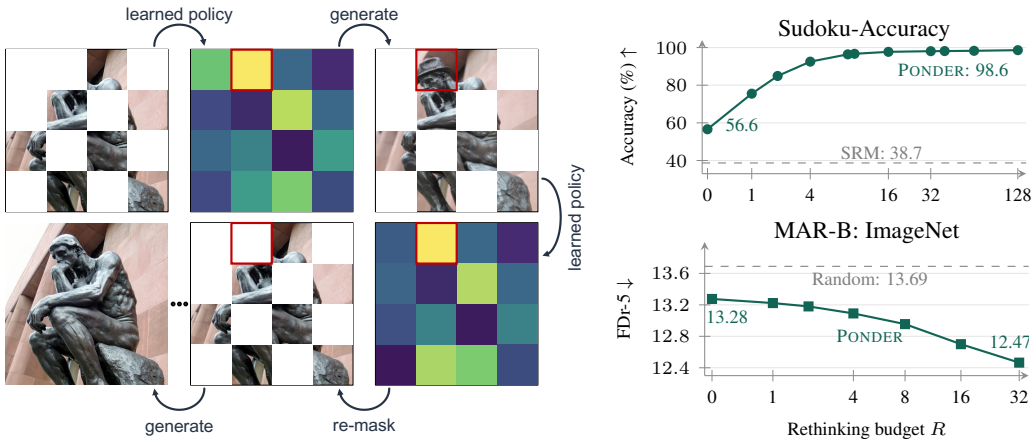
LEARNING TO GENERATE IN THE RIGHT ORDER

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce PONDER, a method that learns in which order an image generator should fill in content and which of its earlier predictions to revise. Masked generative models construct a sample by repeatedly filling in subsets of missing variables. The order of these partial predictions determines the context available in subsequent steps and therefore affects the quality of the final sample. Common strategies are to choose the order randomly or based on local confidence estimates. Conditioned on the partially generated sample and the generator's features, our policy chooses which positions to fill next and/or which previously filled positions to mask for later regeneration, a mechanism we call rethinking. In this way, the generation process is able to adapt to the evolving context and also to revise earlier predictions. We optimize the policy using group-relative policy gradients, with rewards that measure constraint satisfaction, human preference, or distribution quality. For the latter, we introduce FIDgrad, an image-level reward derived from the (first-order) change in Fréchet Inception Distance caused by adding a generated sample to the current feature distribution. With the generators frozen, learned ordering reduces MAR-B ImageNet FID from 4.09 to 2.78 and raises the accuracy on hard visual Sudokus from 3.9% to 56.6%, surpassing uncertainty-based greedy ordering (38.7%). Rethinking further improves these results to 2.70 FID and 96.7% accuracy, while increasing the rethinking budget of a fixed policy yields continued improvements at inference time.



(a) Learning in which order to (re-)generate

(b) Quality scales with rethinking budget

Figure 1: **Learning to generate and rethink.** Left: a learned policy selects positions of patches to generate or mask for later regeneration, enabling revision of earlier predictions. Right: increasing the inference-time rethinking budget improves Sudoku-hard accuracy and MAR-B image quality, with each policy and generator held fixed.

1 INTRODUCTION

Standard diffusion models generate images by iteratively denoising all spatial locations jointly (Ho et al., 2020; Lipman et al., 2023; Karras et al., 2022; Rombach et al., 2022). An alternative is to factorize generation over spatial locations, using a diffusion model to predict individual patches or

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

subsets conditioned on content that has already been generated (Li et al., 2024). In this setting, the generation order becomes consequential because each prediction determines the context available for subsequent ones. It has been shown that such conditional generation is capable of high-quality image synthesis and spatial reasoning (Fan et al., 2025; Wewer et al., 2025), but also that its performance depends substantially on the chosen patch ordering (Fan et al., 2025; Wewer et al., 2025).

If all conditionals were known exactly, different orderings would yield equivalent factorizations. However, the conditionals that are actually learned do not preserve the equivalence. In practice, generation order substantially affects sample quality (Wewer et al., 2025; Li et al., 2024; Fan et al., 2025). This observation leads to a natural question: *Can we learn to adapt the ordering in a sample-dependent way, such that the complete, final sample has the best possible quality?*

There are three common strategies to define the order: *Fixed* schedules, most often in raster-scan order, are independent of the generated content (Esser et al., 2021; Wang et al., 2026; **?**). *Random* schedules sample a permutation per roll-out, as in masked autoregressive models (Li et al., 2024). These decorrelate training from any particular order but provide no mechanism for exploiting context during inference. *Greedy* schedules prioritize positions according to confidence or (predicted) difficulty: MaskGIT (Chang et al., 2022; Li et al., 2026) unmasks the tokens the generator is most confident about, SRM (Wewer et al., 2025) selects the ones with the lowest predicted uncertainty, and Patch Forcing (Schusterbauer et al., 2026) estimates a per-patch difficulty score and advances easy regions first, so as to obtain more context for harder ones. All three families share the same limitation: either the order of generation is completely independent of the sample being generated, or it is determined by a greedy one-step proxy.

Among those strategies, only greedy confidence-based schedules adapt the ordering to the intermediate, partial state at all. However, the decision is based on an extremely local proxy: it prioritizes positions that are easy to predict given the current context. Such a strategy does not consider how complex future steps will become based on that decision. Thus, a local prediction that is made with high confidence does not, by itself, mean that it will greatly improve the global sample quality. A highly predictable position may provide little useful context, whereas a less certain one may establish useful structure that constrains further patches.

The delayed effect of intermediate decisions turns generative modeling into a sequential decision-making problem, driven by a policy that selects positions conditioned on the current partial sample. Taking that view further, earlier predictions made with limited context may become inconsistent at later stages of the generation process, calling for a mechanism to also revisit earlier choices.

We introduce **PONDER** (Policy for Ordering, Next-element DEcisions and Rethinking), a scheme for learning a policy that controls the sequence order during generation, including removal of earlier predictions for subsequent regeneration. To that end, we make the following contributions:

- *Learned ordering for continuous generators.* A policy trained with final-sample rewards that decides where a frozen generator fills content next. It requires only the policy's own action probabilities, not the generator's likelihoods, which makes it the first such approach applicable to continuous-token generators with diffusion heads.
- A *rethinking* strategy, where the policy learns which earlier predictions to discard and regenerate as new context becomes available. This enables adaptive revision during sequential generation with a controllable rethinking budget that can be varied at inference time.
- *FIDgrad*, a reward function that turns a distribution-level objective into an image-level reinforcement learning signal. By estimating each sample's first-order effect on Fréchet Inception Distance, FIDgrad enables a generation policy that directly targets quality at the distribution level.

A suite of experiments demonstrates the generality of PONDER across class-conditional generation, text-to-image synthesis, and visual constraint solving. With learned ordering alone, it reduces MAR-B ImageNet FID from 4.09 to 2.78, improves MaskGen-KL-XL HPSv2 from 24.50 to 26.50, and raises hard-Sudoku accuracy from 3.9% to 56.6%. Rethinking further improves these results, to 2.70 FID, 27.80 HPSv2 at $R = 32$, and 96.7% accuracy. Moreover, performance continues to improve as the inference-time budget increases while the policy and generator remain fixed (Figure 1).

2 Related Work

Masked generation with continuous representations. Masked generative models predict missing content from a partially observed sample, allowing generation to proceed in different orders autoregressively. MaskGIT (Chang et al., 2022) uses discrete image tokens and iterative confidence-based decoding. MAR (Li et al., 2024) instead models conditional distributions over continuous patches using a patch-wise diffusion head, removing the need for vector quantization. FLUID (Fan et al., 2025) studies the effects of token representation and generation order in text-to-image synthesis, finding advantages for continuous tokens and random-order generation. MaskGen (Kim et al., 2025) supports both discrete and continuous representations through compact one-dimensional image tokens. We build on continuous masked generators and learn how to use their conditional predictions while keeping their weights frozen.

Adaptive and learned generation orders. SRM (Wewer et al., 2025) demonstrates the importance of sequentialization and generation order for reasoning over continuous spatial variables, using predicted uncertainty to select the next position. Patch Forcing (Schusterbauer et al., 2026) learns per-patch difficulty estimates and adapts denoising progress across an image, allowing easier regions to provide context for harder ones. These approaches motivate adapting generation to the current sample. Our policy is trained against rewards on the completed image. Learned ordering has also been studied in discrete diffusion language models. DCoLT (Huang et al., 2025) uses final-answer rewards to optimize the reverse generation trajectory and its LLaDA generator jointly trains a token-unmasking model and the language model, adapting both generation order and token predictions. **?** isolate the ordering component by keeping the language model frozen and learning a position-selection policy from verifiable terminal rewards, with regularization relative to confidence-based reference policies. Fox et al. (2026) approach ordering through variational inference: a posterior conditioned on complete training examples provides a learning signal for an inference-time token selector through an evidence lower bound. Concurrent work by Ouyang et al. (2026) learns an unmasking controller with group-relative policy optimization for multimodal diffusion language models. All of these methods operate on discrete tokens and only decide which masked position to reveal next; once revealed, a token stays fixed. Several also rely on quantities that continuous generators with a diffusion head do not provide, such as token confidences or likelihoods. Our work studies continuous-latent diffusion generators and additionally learns which previously generated positions to remove and later regenerate under a rethinking budget.

Revision during generation. Allowing accepted predictions to change can correct errors that become apparent as additional context is generated. Path Planning (Peng et al., 2025) augments masked discrete diffusion sampling with planners that can select both masked and previously unmasked tokens for updating. It also derives an evidence lower bound for training planners with a frozen denoiser. Nemotron-Labs-Diffusion-Image (Li et al., 2026) enables revision of discrete image tokens through an augmented corruption objective and confidence-thresholded token replacement at inference. PONDER learns revision decisions from final-sample rewards without retraining the generator. Its position-selection policy allocates a prescribed removal budget: removed content is excluded from subsequent context and regenerated when its position is selected again. This separates the decision to discard a prediction from its later regeneration.

Reward optimization for generative models. DDPO (Black et al., 2024) formulates diffusion denoising as a sequential decision process and optimizes generator parameters using downstream rewards. Flow-GRPO (Liu et al., 2025) extends group-relative policy optimization to flow matching models, while Mask-GRPO (Luo et al., 2025) develops reward-based post-training for masked text-to-image generators. PONDER applies reinforcement learning to the position-selection policy, leaving content generation to the frozen model. The optimized action probabilities are those of selecting positions to fill or remove, so policy training does not require evaluating the likelihood of generated continuous content.

Distributional objectives. FD-loss (Yang et al., 2026) optimizes representation-space Fréchet distance by maintaining a large feature population through a queue or moving moments while back-propagating through the current generated batch. FIDgrad shares the use of evolving population statistics but derives a scalar reward reflecting the incfluence of a single sample on the FID. It

measures the first-order effect of adding probability mass at a generated image's feature. This reward trains the position-selection policy through policy gradients, without differentiating through the generator or feature extractor. It therefore connects a distributional quality objective to discrete generation and revision decisions over continuous content.

3 METHOD

Sequential image generation can be cast as masked image modeling. Starting from an empty canvas, the generator unmasks one or multiple regions (in practice: patches) and fills them by generating suitable pixel values or latents. That process is repeated in autoregressive fashion until the entire canvas has been filled. The goal of PONDER is to devise a policy in which order to unmask the patches. Additionally, the policy can also choose to re-mask patches, such that they can be regenerated in subsequent steps. The policy is trained to maximize a suitable target metric for the task at hand, e.g., FID (Heusel et al., 2017b) or Human Preference Score (Wu et al., 2023b).

3.1 POLICY-GUIDED MASKED GENERATION

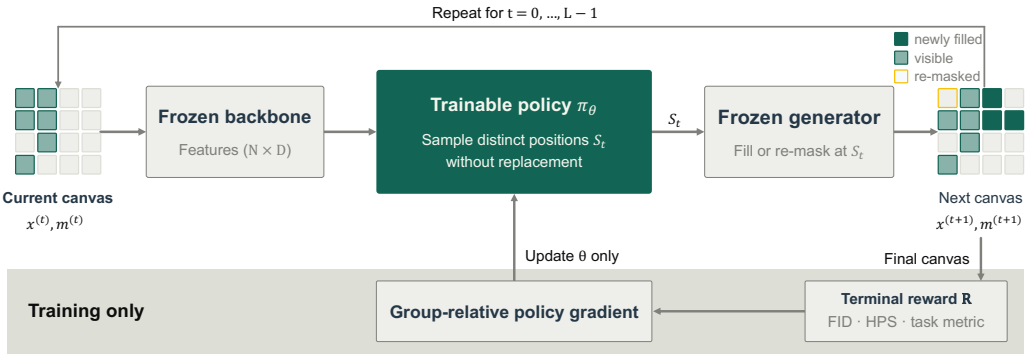


Figure 2: **Overview of PONDER.** The policy processes frozen backbone features using transformer blocks and a linear head, then samples K_t positions without replacement. Selected masked positions are filled, while rethinking masks selected visible positions for later regeneration. A terminal reward trains only the policy through group-relative policy gradients, while the generator remains frozen.

Image generation proceeds for L iterations indexed by $t = 0, \ldots, L-1$. Each iteration consists of three steps: evaluating the generator backbone for the current canvas $x^{(t)}$, selecting patches to unmask, and updating the canvas by generating content for the unmasked patches. At each iteration, the input to the backbone are image tokens for the patches that are already visible and MASK tokens for the patches that are still masked. The backbone is evaluated once and computes per-patch features that remain fixed throughout the iteration and constitute the input $h^{(t)}$ for the policy. For diffusion-based approaches, the backbone features are concatenated with the initial noise. Note that the iterations of the sequential generator are different from the "inner loop" of denoising steps during diffusion.

The goal of the generation process is to fill all writeable patches $\mathcal{W} \subseteq \{1, \ldots, N\}$. A prescribed schedule determines the number K_t of patches to be unmasked in each iteration, while the policy chooses their locations S_t. A binary visibility mask $m^{(t)} \in \{0,1\}^N$ indicates which positions are already visible as context. Optionally, there may be additional class or text conditioning c. Using the context computed at the start of the iteration, the generator fills the selected patches and the mask is updated to the new state:

$$\begin{aligned} x_{S_t}^{(t+1)} &= g_\phi\big(x^{(t)}, S_t, \epsilon_{S_t}^{(t)}, m^{(t)}, c\big), \\ m_{S_t}^{(t+1)} &= \neg m_{S_t}^{(t)} \end{aligned} \tag{1}$$

where g_ϕ denotes the generator and $\epsilon_{S_t}^{(t)}$ is the noise initialization at the selected positions. Moreover, we extend the policy with a *rethinking* mode. Formally, the selection S_t may contain already visible patches ($m_i^{(t)} = 1$), which are then reverted back to the masked state ($m_i^{(t+1)} = 0$).

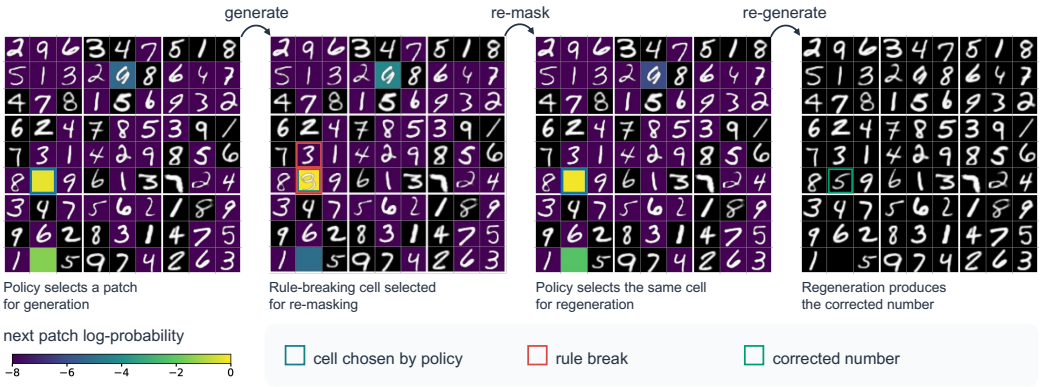


Figure 3: An example of sequential generation with rethinking on a visual Sudoku instance. We visualize next patch log probabilities as background color and policy decisions as border colors.

3.2 Adaptive Patch Selection

The policy π_θ is a bidirectional transformer that maps the generator features $h^{(t)}$ and start noise vectors to output logits. The policy obtains the same information that the patch-wise diffusion heads obtain. We ablate this choice in Table 5. The output logits consist of K_t sets of per-patch logits, one for each decision to be taken in the current iteration (Fig. 2). Patches are sampled sequentially, excluding those that have already been selected in the same iteration.

Let $\mathcal{C}_{t,k}$ denote the candidate patches for decision k. The probability of the ordered selections is

$$\pi_\theta(a_{t,1:K_t} \mid h^{(t)}) = \prod_{k=1}^{K_t} \pi_{\theta,k}(a_{t,k} \mid h^{(t)}, \mathcal{C}_{t,k}). \tag{2}$$

All decisions within an iteration share the same generator features. Earlier selections affect later ones only through the candidate mask. After selection, the generator fills $S_t = \{a_{t,1}, \ldots, a_{t,K_t}\}$ using the same context from the previous iteration for all of them. Without rethinking, the complete sequence to a complete output visits every writable patch exactly once.

3.3 Pondering by Revisiting Patches

Restricting the generation order to strict permutations simplifies the process, but makes it impossible to correct early decisions after additional context becomes available. To enable such revisions, we allow the policy to remove previously generated content. We note that the same patch selection policy can control both generation and revision. Only a minor change is needed, namely to allow the selection of already visible patches. Whenever a masked patch is selected in iteration t, it is filled. When an unmasked patch is selected, it is returned to the masked state, thus excluding it from the backbone evaluation in the next iteration $t + 1$. The patch then remains masked until the policy selects it again, at which point it is regenerated using the new context.

Because backbone features are computed before position selection, removal does not change the features used within the current round and requires no additional backbone evaluation in that round. The next scheduled backbone evaluation uses the updated mask.

We prescribe a rethinking budget of R removal actions. Starting from a fully masked canvas, completing the sample requires N net fill actions. Each removal cancels one previous fill and therefore requires one additional fill later in the roll-out. A trajectory with R removals consequently contains $N + R$ fill actions and R removal actions, for a total of

$$T = N + 2R \tag{3}$$

patch decisions. A patch may be removed more than once, with each removal counting toward the budget. Book-keeping of legal decisions ensures completion: removals are disabled once the budget is exhausted and in the final iteration $L - 1$. Conversely, they are forced when necessary to exhaust the budget before the final iteration.

Importantly, rethinking is distinct from repeatedly resampling a position under the same context, because regeneration can happen after other parts of the canvas have been filled, providing more context for the generator. The policy therefore learns not only where to generate next, but also when an earlier prediction is worth revisiting. This differs from fixed refinement schedules that allocate additional computation independently of the evolving sample.

Setting $R = 0$ recovers adaptive ordering without rethinking. Because R changes the roll-out horizon but not the frozen generator or the policy parameterization, it can be varied at inference time within the schedules supported by the policy.

3.4 Group-Relative Policy Optimization

We train the ordering policy from a terminal reward $\mathcal{R}(x^{(L)}, c)$ evaluated on the completed sample. For each input and conditioning value, we generate a group of G trajectories. Members of a group share the same initial canvas and generator noise, but sample their policy actions independently, to isolate the effect of the generation order on the final image and the associated reward. Let r_j denote the terminal reward of trajectory j, and let $\bar{r}$ and s denote the group mean and population standard deviation. We assign every decision in trajectory j the advantage

$$A_j = \begin{cases} (r_j - \bar{r})/s, & s > 10^{-5}, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Index the $T = N + 2R$ primitive position decisions by d. Let o_{jd} contain the policy observation and legal candidate set at decision d of trajectory j. At optimizer update u, we minimize

$$\mathcal{L}_u(\theta) = -\frac{1}{GT} \sum_{j=1}^{G} \sum_{d=1}^{T} \left[\mathrm{sg}(A_j) \log \pi_\theta(a_{jd} \mid o_{jd}) \right], \tag{5}$$

Here sg stops gradients. We average this loss across independently collected groups.

Gradients update only the ordering policy, whereas the generator remains frozen. Batching several independent groups averages the loss across groups. We use a single policy-gradient update per collected roll-out batch, without reference-policy penalty, importance ratios, clipping, or repeated optimization over the same roll-out. Further training details are given in Appendix A.1.

3.5 Distribution-Level Optimization with FIDGrad

Standard rewards, like constraint satisfaction or human preference, can be evaluated independently for each generated sample. In contrast, population-level quality metrics, like the Fréchet Inception Distance (FID), measure the discrepancy between the distributions of generated and real images and cannot directly be broken down into per-sample rewards. Inspired by the FD-loss (Yang et al., 2026) for generative model training, we introduce *FIDgrad*: a reward that scores each generated image by its first-order effect on the FID of the current generative distribution. This makes it possible to optimize distribution quality through reinforcement learning, while keeping the generative model frozen.

Let q denote the distribution of Inception features $z = f(x)$ of the generated images, with mean μ_g and covariance Σ_g. And let μ_r, Σ_r be the corresponding statistics of the real image data. We work with the standard Gaussian plug-in functional, which depends on q only through its first two moments,

$$\mathrm{FID}(q, p_{\mathrm{real}}) = \|\mu_g - \mu_r\|_2^2 + \mathrm{tr}(\Sigma_g) + \mathrm{tr}(\Sigma_r) - 2\,\mathrm{tr}\Big[\big(\Sigma_r^{1/2}\Sigma_g\Sigma_r^{1/2}\big)^{1/2}\Big]. \tag{6}$$

To turn Eq. 6 into a per-sample quantity, we ask how it responds to placing a small amount of probability mass at a single feature z, i.e., we evaluate the influence function of the functional at q.

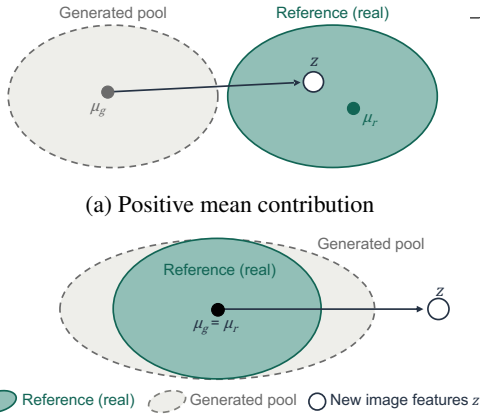
Proposition 1 (FIDgrad). *Assume $\Sigma_g \succ 0$ and $\Sigma_r \succ 0$. For a feature z, let $q_\epsilon = (1-\epsilon)q + \epsilon\delta_z$ with δ_z a point mass at z, and let $u = z - \mu_g$. Then $\epsilon \mapsto \mathrm{FID}(q_\epsilon, p_{\mathrm{real}})$ is differentiable at $\epsilon = 0$ and*

$$r_{\mathrm{FIDgrad}}(z) := -\tfrac{\mathrm{d}}{\mathrm{d}\epsilon}\,\mathrm{FID}(q_\epsilon, p_{\mathrm{real}})\big|_{\epsilon=0} = -2(\mu_g - \mu_r)^\top u - u^\top (I - B)u + \mathrm{tr}[(I-B)\Sigma_g], \tag{7}$$

where $B = \Sigma_r^{1/2}\big(\Sigma_r^{1/2}\Sigma_g\Sigma_r^{1/2}\big)^{-1/2}\Sigma_r^{1/2}$ is symmetric positive definite, and $B = I$ if and only if $\Sigma_g = \Sigma_r$.

A positive reward identifies samples whose inclusion would decrease FID to first order. The first term captures the contribution of a sample to correcting the mean of the generated features, the second term captures its contribution to correcting the feature covariance. Figure 4 illustrates the resulting effect.

FIDgrad depends on both the current generated distribution and the individual image, so the same image can receive different rewards as the policy evolves. During training, we track the generated mean and covariance using exponential moving first and second moments. Each batch is scored before these estimates are updated, while the real-data moments remain fixed. Initialization and numerical conventions are specified in Appendix A.1.2. The resulting scores serve as terminal rewards in the group-relative objective.



(a) Positive mean contribution

(b) Negative covariance contribution

Figure 4: Illustration of the mean and covariance component of the FIDgrad reward for a new image with features z. FID compares Gaussian approximations of generated and reference image distributions in Inception feature space (Heusel et al., 2017a). Our reward measures the first order change in FID induced by adding the new image into the pool of generated images. a) The new sample z shifts the mean of the generated distribution towards the reference distribution. This contributes to a positive reward. b) The new sample increases the variance of the generated distribution, although the variance is already higher than the reference distribution. This contributes negatively to the reward. In general a sample affects both: mean and covariance.

4 EXPERIMENTS

We train and evaluate PONDER policies for three generative models: MAR-B (ImageNet, Li et al., 2024), MAR (Sudoku; similar to SRM, Wewer et al., 2025), and the continuous-token text-to-image model MaskGen-KL-XL (Kim et al., 2025). All generators are frozen throughout our experiments. To speed up policy training, we use DDIM sampling with 25 steps for MaskGen-KL-XL and MAR-B, and flow matching (Lipman et al., 2023) with 12 steps for the Sudoku network. Policy training details are described in Appendix A.1, and generator architectures and inference settings in Appendix A.2. To our knowledge, PONDER is the first method to learn generation and revision order for continuous-token generators, so no prior method applies directly. We therefore construct baselines that isolate each component.

Visual Sudoku. Following SRM (Wewer et al., 2025), we generate images of handwritten (MNIST) Sudokus. Such samples can look realistic while violating the Sudoku rules, making the task a test of global constraint satisfaction rather than local image quality. We classify every cell and verify the row, column, and box constraints, using SRM's classifier and its *easy*, *medium*, and *hard* splits, which differ in the number of given cells. The frozen generator is an MAR (Li et al., 2024) with a flow-matching head operating on per-cell VAE latents (Appendix A.2, Table 7). A single policy is trained for all difficulty levels with a binary reward for a valid solution (Appendix A.1.4).

Learned ordering alone raises hard-split accuracy from 3.9% to 56.6%, well above SRM's uncertainty-based greedy ordering (38.7%, Table 1); with a rethinking budget of $R = 9$, it reaches 96.7%. The resampling of patches itself only helps marginally: with random ordering, $R = 9$ leaves accuracy at 4.0%, and with the learned order but randomly chosen removal locations, it reaches only 62.7% compared to 56.6% with no rethinking and 96.7% with learned order and learned rethinking positions. The 34-percent gap to learned rethinking is therefore attributable to learning *which* predictions to revisit. The policy trained with $R = 9$ also generalizes to larger test-time budgets, reaching 98.6% at $R = 128$ (Figure 5).

ImageNet (MAR). We apply PONDER to class-conditional ImageNet generation with the frozen MAR-B generator, training the policy with FIDgrad (Sec. 3.5). Since FIDgrad is computed in Inception feature space, we additionally report FDr-5 to check against reward hacking. It follows the

MAR-Sudoku					
Position policy	R	Overall ↑	Hard ↑	Medium ↑	Easy ↑
Random order	0	50.5	3.9	51.5	96.1
SRM greedy	0	70.2	38.7	75.7	96.3
Random order	9	51.7	4.0	55.1	96.2
PONDER	0	80.2	56.6	87.1	96.9
PONDER w. random rethinking	9	83.1	62.7	89.0	97.6
PONDER	9	**98.2**	**96.7**	**98.4**	**99.6**

Table 1: Exact-solution accuracy (%) on MNIST Sudoku. R is the rethinking budget. The two $R = 9$ PONDER rows retain learned order and the same removal budget, varying only the selection of removal locations.

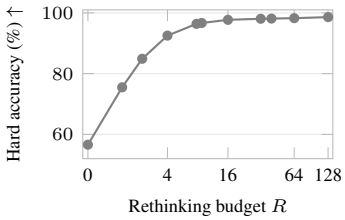


Figure 5: Scaling the rethinking budget at test time improves accuracy dramatically.

ImageNet: MAR-B				
Position policy	R	FID ↓	FDr-5 ↓	IS ↑
Random	0	4.09	13.69	181.22
Random	32	3.80	12.82	185.90
PONDER	0	2.78	13.28	206.55
PONDER w. random rethinking	32	2.76	12.68	209.00
PONDER	32	**2.70**	**12.47**	**212.60**

Table 2: ImageNet-256 results with frozen MAR-B.

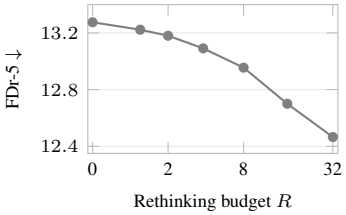


Figure 6: MAR-B test-time scaling; FDr-5 is not used as reward.

FDr-6 metric of Yang et al. (2026) but excludes the Inception backbone, and thus measures Fréchet distance in five representation spaces that are not used by the reward (Appendix A.4). Without rethinking, PONDER reduces FID from 4.09 to 2.78 and increases IS from 181.22 to 206.55 compared with random ordering. FDr-5 improves as well (13.69 → 13.28). Increasing the test-time rethinking budget of the same policy to $R = 32$ further improves all three metrics, reaching 2.70 FID, 12.47 FDr-5, and 212.60 IS (Figure 6). Additional budgets are reported in Appendix Table 9. Figure 6 plots the corresponding FDr-5 trend.

Text-to-image generation. We train two separate ordering policies for the frozen MaskGen-KL-XL generator. One uses Human-Preference-Score v2.1 (Wu et al., 2023a) (HPSv2) as reward on HPSv2 training prompts. We train it with a uniformly sampled rethinking budget from $R = 0$ to $R = 8$; each update uses one sampled budget for all trajectories. We evaluate on the official HPSv2 benchmark prompts and additionally report PickScore (Kirstain et al., 2023) and ImageReward (Xu et al., 2023). At $R = 0$, PONDER scores 26.50 HPSv2 versus 24.50 for random ordering in a separate evaluation. Increasing R to 4 with the same policy raises HPSv2 to 26.95 and ImageReward from 0.68 to 0.73. At $R = 32$, HPSv2 reaches 27.80 and ImageReward reaches 0.81 (Figure 8). We train a separate policy with the same $R \in \{0, \ldots, 8\}$ mixture on the GenEval-style training prompts released by Flow-GRPO (Liu et al., 2025), using its dense partial-credit reward. We report the original binary GenEval metric (Ghosh et al., 2023). Results are shown in Tables 3 and 4. On

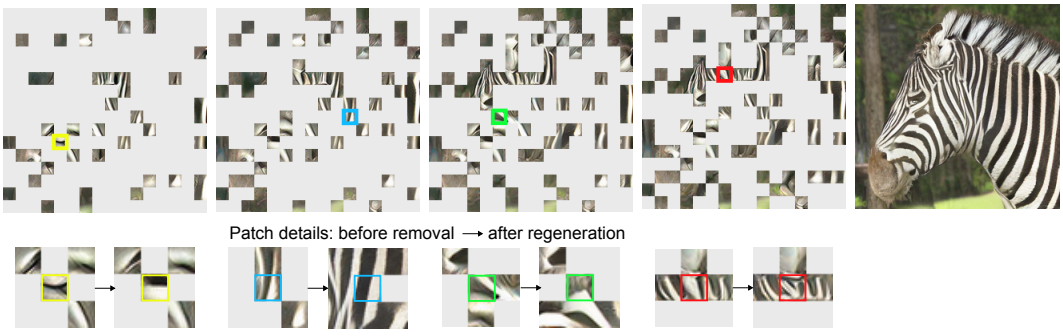


Figure 7: We visualize the successive generation of a MAR-B image. In the bottom row we visualize patches before they have been remasked and after after the same patch has been re-generated.

MaskGen-KL-XL				
Position policy	R	HPSv2 ↑	PickScore ↑	ImageReward ↑
Random	0	24.50	21.04	0.44
Random	4	25.19	21.26	0.59
PONDER	0	26.50	21.32	0.68
PONDER w. random rethinking	4	26.40	21.33	0.67
PONDER	4	**26.95**	**21.41**	**0.73**

Table 3: MaskGen-KL-XL preference scores on the official HPSv2 prompts.

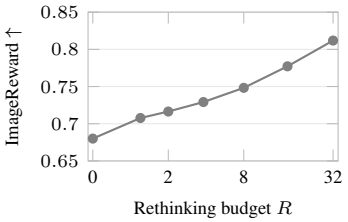


Figure 8: MaskGen test time scaling for the HPSv2 policy.

MaskGen-KL-XL								
Policy	R	Overall ↑	Single ↑	Two ↑	Count ↑	Color ↑	Pos. ↑	Attr. ↑
Random	0	37.3	88.4	32.1	21.3	66.5	5.0	10.8
Random	4	38.0	89.7	37.6	21.6	63.8	5.3	9.8
PONDER	0	41.1	93.4	35.1	30.9	**67.8**	6.8	12.5
PONDER w. random rethinking	4	41.9	**95.0**	**39.1**	29.7	65.2	**7.5**	15.0
PONDER	4	**42.6**	**95.0**	38.6	**31.9**	67.0	**7.5**	**15.5**

Table 4: MaskGen-KL-XL results on the official GenEval prompts . Scores are percentages.

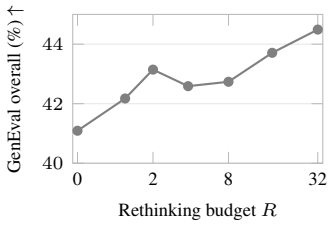


Figure 9: MaskGen test time scaling for the GenEval policy.

GenEval, overall accuracy rises from 37.3% with random ordering to 41.1% with learned ordering at $R = 0$. At $R = 4$, the learned policy scores 42.6% versus 38.0% for random ordering (Table 4). Figure 9 shows the learned-policy test-time scaling trend.

Table 5: Policy-input ablation with frozen MaskGen-KL-L and MAR-Sudoku. The benefit of observing generation noise depends on the generator and task: it helps MaskGen in this comparison, but is not required for strong Sudoku performance. Generator features remain useful in both settings.

	MaskGen-KL-L HPSv2 ↑		MAR-Sudoku ($R = 9$) Accuracy (%) ↑			
Policy input	$R = 0$	$R = 4$	Overall	Hard	Medium	Easy
Generator features + noise	**26.11**	**26.61**	96.38	95.27	96.67	**97.21**
Generator features only	25.01	25.71	**96.49**	**95.72**	**96.78**	96.97
Latent state only	25.05	25.42	59.19	22.11	59.82	95.64

5 CONCLUSION AND DISCUSSION

We present PONDER, a general framework for learning the generation order of frozen masked generative models. Rather than relying on fixed, random, or locally greedy schedules, PONDER learns a position-selection policy from rewards defined on the completed sample. The same policy can also revisit earlier predictions through *rethinking*, allowing decisions made with limited context to be removed and reconsidered once additional information becomes available. Across visual constraint solving, class-conditional image generation, and text-to-image synthesis, this improves the observed metrics over the generator's default sampling schedule, while leaving the underlying generator untouched. Importantly, improvements are not limited to the reward we used. This suggests that the observed gains are not solely due to overoptimization of the training reward. Generation order thus becomes an optimizable component of the generative process rather than a predefined heuristic.

Limitations. The policy adds a forward pass per generation iteration, which is small relative to the generative backbone and the scheduled denoising steps but adds a minor overhead. Its behavior is strongly dependent on the reward that is used for training. Reward design therefore remains a critical requirement for policy success. Finally, rethinking is governed by a prescribed budget and not allocated on demand. Learning *when* to stop, in addition to where to generate and revise, and extending the action space to inference parameters such as guidance strength, are natural next steps toward generative models that learn their own inference strategy.

AI USE STATEMENT

We used generative AI tools to assist with implementing and debugging research code, provide feedback on experimental methodology, identify relevant literature, assist with the mathematical derivations for the FIDgrad reward and revise the manuscript. The authors reviewed and revised the AI-assisted text, checked references against the original sources, checked the mathematical derivations step by step under the stated assumptions and verified the resulting reward expression and reviewed and tested AI-assisted code. We take responsibility for the final manuscript, methods, results, and accompanying code, including all content produced with generative AI assistance.

REFERENCES

Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. In *International Conference on Learning Representations*, 2024.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. MaskGIT: Masked generative image transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11305–11315, 2022. doi: 10.1109/CVPR52688.2022.01103.

Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12868–12878, 2021. doi: 10.1109/CVPR46437.2021.01268.

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. In *International Conference on Learning Representations*, 2025.

David Fox, Sam Bowyer, Song Liu, Laurence Aitchison, Raul Santos-Rodriguez, and Mengyue Yang. Learning generation orders for masked discrete diffusion models via variational inference. *arXiv preprint arXiv:2602.23968*, 2026.

Dhruba Ghosh, Hannaneh Hajishirzi, and Ludwig Schmidt. GenEval: An object-focused framework for evaluating text-to-image alignment. In *Advances in Neural Information Processing Systems 36*, pp. 52132–52152, 2023. doi: 10.52202/075280-2270.

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 *Advances in Neural Information Processing Systems 30*, pp. 6626–6637, 2017a.

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 *Advances in neural information processing systems 30*, volume 30, 2017b.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems 33*, pp. 6840–6851, 2020.

Zemin Huang, Zhiyang Chen, Zijun Wang, Tiancheng Li, and Guo-Jun Qi. Reinforcing the diffusion chain of lateral thought with diffusion language models. In *Advances in Neural Information Processing Systems 38*, pp. 169486–169519, 2025. doi: 10.52202/085713-5105.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems 35*, pp. 26565–26577, 2022. doi: 10.52202/068431-1926.

Dongwon Kim, Ju He, Qihang Yu, Chenglin Yang, Xiaohui Shen, Suha Kwak, and Liang-Chieh Chen. Democratizing text-to-image masked generative models with compact text-aware one-dimensional tokens. In *IEEE/CVF International Conference on Computer Vision*, pp. 18442–18452, 2025. doi: 10.1109/ICCV51701.2025.01714.

Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. Pick-a-Pic: An open dataset of user preferences for text-to-image generation. In *Advances in Neural Information Processing Systems 36*, pp. 36652–36663, 2023. doi: 10.52202/075280-1594.

Shufan Li, Greg Heinrich, Hanrong Ye, Yonggan Fu, Aditya Grover, Jan Kautz, and Pavlo Molchanov. Nemotron-Labs-Diffusion-Image: Advancing masked discrete diffusion for high-resolution image synthesis. *arXiv preprint arXiv:2606.29814*, 2026.

Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. In *Advances in Neural Information Processing Systems 37*, pp. 56424–56445, 2024. doi: 10.52202/079017-1797.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-GRPO: Training flow matching models via online RL. In *Advances in Neural Information Processing Systems 38*, pp. 45650–45685, 2025. doi: 10.52202/085713-1362.

Yifu Luo, Xinhao Hu, Keyu Fan, Haoyuan Sun, Zeyu Chen, Bo Xia, Tiantian Zhang, Yongzhe Chang, and Xueqian Wang. Reinforcement learning meets masked generative models: Mask-GRPO for text-to-image generation. In *Advances in Neural Information Processing Systems 38*, pp. 120047–120072, 2025. doi: 10.52202/085713-3616.

Yidong Ouyang, Zhe Wang, Sourav Bhabesh, and Dmitriy Bespalov. Reinforcing the generation order of multimodal masked diffusion models. *arXiv preprint arXiv:2607.08056*, 2026.

Fred Zhangzhi Peng, Zachary Bezemek, Sawan Patel, Jarrid Rector-Brooks, Sherwood Yao, Avishek Joey Bose, Alexander Tong, and Pranam Chatterjee. Path planning for masked diffusion model sampling. *arXiv preprint arXiv:2502.03540*, 2025.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10674–10685, 2022. doi: 10.1109/CVPR52688.2022.01042.

Johannes Schusterbauer, Ming Gui, Yusong Li, Pingchuan Ma, Felix Krause, and Björn Ommer. Denoising, fast and slow: Difficulty-aware adaptive sampling for image generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Xinlong Wang, Yufeng Cui, Jinsheng Wang, Fan Zhang, Yueze Wang, Xiaosong Zhang, Zhengxiong Luo, Quan Sun, Zhen Li, Yuqi Wang, Qiying Yu, Yingli Zhao, Yulong Ao, Xuebin Min, Chunlei Men, Boya Wu, Bo Zhao, Bowen Zhang, Liangdong Wang, Guang Liu, Zheqi He, Xi Yang, Jingjing Liu, Yonghua Lin, Zhongyuan Wang, and Tiejun Huang. Multimodal learning with next-token prediction for large multimodal models. *Nature*, 650(8101):327–333, 2026. doi: 10.1038/s41586-025-10041-x.

Christopher Wewer, Bartlomiej Pogodzinski, Bernt Schiele, and Jan Eric Lenssen. Spatial reasoning with denoising models. In *International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 66706–66725, 2025.

Xiaoshi Wu, Yiming Hao, Keqiang Sun, Yixiong Chen, Feng Zhu, Rui Zhao, and Hongsheng Li. Human Preference Score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. *arXiv preprint arXiv:2306.09341*, 2023a.

Xiaoshi Wu, Keqiang Sun, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score: Better aligning text-to-image models with human preference. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2096–2105. IEEE, 2023b.

Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. ImageReward: Learning and evaluating human preferences for text-to-image generation. In *Advances in Neural Information Processing Systems 36*, pp. 15903–15935, 2023. doi: 10.52202/075280-0700.

Jiawei Yang, Zhengyang Geng, Xuan Ju, Yonglong Tian, and Yue Wang. Representation Fréchet loss for visual generation. *arXiv preprint arXiv:2604.28190*, 2026.

A APPENDIX

A.1 POLICY TRAINING DETAILS

A.1.1 SHARED SETUP

The generators remain frozen while we train their ordering policies.

Trajectories within a conditioning group share generation noise, indexed by position and the number of previous fills. The policy observes the noise for each position's next fill, while policy actions are sampled using a separate random stream. Evaluations use matched noise across rethinking budgets, with the noise capacity held fixed.

All ordering policies are bidirectional pre-norm Transformer encoders with GELU activations, learned positional embeddings, and zero dropout. Unless otherwise noted, each position's input concatenates the frozen generator feature with the Gaussian noise for the next fill. Table 6 reports the trainable policy parameters and optimization settings.

Table 6: Dynamic-removal policy architectures and optimization settings.

Parameter	MAR-B	MaskGen-KL-XL	MAR-Sudoku
Architecture			
Number of positions	256	32	81
Generator feature width	768	1280	256
Noise width	16	16	32
Input projection	$784 \to 768$	$1296 \to 1024$	$288 \to 256$
Transformer depth	6	4	2
Model width	768	1024	256
Attention heads	8	8	4
Feed-forward width	3072	4096	1024
Action heads	7	1	2
Trainable parameters	43.334M	51.749M	1.675M
Optimization			
Optimizer	Adam	Adam	Adam
Learning rate	2×10^{-5}	2×10^{-5}	2×10^{-5}
Learning-rate schedule	constant, then linear decay	constant	constant
Adam (β_1, β_2)	$(0.9, 0.95)$	$(0.9, 0.95)$	$(0.9, 0.95)$
Maximum training updates	50k	50k	50k
Conditioning inputs/update	128	128	128
Trajectories per group	8	8	16
Trajectories per update	1024	1024	2048
Gradient-norm clip	1.0	1.0	1.0

All policies use Adam with $\epsilon = 10^{-8}$ and zero weight decay. During policy training, both the policy and frozen generator use bfloat16 mixed precision. Each update contains 128 independent conditioning inputs, each expanded into one group of trajectories. Evaluation precision is reported with each experiment.

A.1.2 MAR-B

We train the MAR-B policy with the FIDgrad reward and a rethinking budget of $R = 8$. The learning rate is held constant for 2,000 updates and then decays linearly to 2×10^{-6} over 2,000 updates.

At update u, let (μ_u, Σ_u) denote the generated feature moments and (μ_r, Σ_r) the fixed real moments. For $d = z - \mu_u$, the perturbation $q_\eta = (1-\eta)q + \eta\delta_z$ changes the mean and covariance to first order by d and $dd^\top - \Sigma_u$, respectively. Combining these with the derivatives of FID gives the reward in Eq. 7 under the assumptions of Proposition 1.

Estimated moments. We maintain the generated mean μ_u and raw second moment M_u, with $\Sigma_u = \mathrm{sym}(M_u - \mu_u \mu_u^\top)$. Every trajectory is scored using the current moments before they are updated. One trajectory is selected uniformly from each group of eight to estimate the batch moments. For selected features $z_1, \ldots, z_n$, the updates are

$$\hat{\mu}_u = n^{-1} \sum_{i=1}^{n} z_i, \qquad \hat{M}_u = n^{-1} \sum_{i=1}^{n} z_i z_i^\top, \tag{8}$$

$$\mu_{u+1} = (1-\alpha)\mu_u + \alpha \hat{\mu}_u, \qquad M_{u+1} = (1-\alpha) M_u + \alpha \hat{M}_u, \tag{9}$$

with $\alpha = 0.001$. The moments are initialized from 50,048 separately generated images. Rewards use 2048-dimensional Inception features from the FD-Loss implementation (Yang et al., 2026) and fixed ImageNet-256 real reference statistics. Moment and matrix computations use double precision. Inverse square roots are computed by eigendecomposition, using a pseudoinverse on numerically singular directions.

Interpretation of the update. The moving moment estimates and within-group reward normalization yield a surrogate objective for policy training. The resulting updates need not decrease population FID at every step.

A.1.3 MaskGen-KL-XL

We train separate policies against HPSv2.1 and the dense GenEval reward, using the same architecture and optimization settings in Table 6. The 553 official GenEval prompts (four images per prompt) are disjoint from the Flow-GRPO training prompts. We report the original binary GenEval score on these prompts.

A.1.4 MAR-Sudoku

We train the MAR-Sudoku policy with a rethinking budget of $R = 9$ and a terminal reward of 1 for a correct completed Sudoku and 0 otherwise. Training starts from partially filled grids. The number of given cells follows a triangular distribution with maximum density at zero and zero density at 81; their locations are sampled randomly.

A.2 Generator architectures and inference settings

Table 7 summarizes the generator architectures and inference settings. Generator parameter counts include the contextual backbone and shared generative head, but exclude tokenizers/VAEs, the MaskGen text encoder, and the Sudoku reward classifier. Both MaskGen-KL-XL policies use the architecture shown here.

For MAR-B, let C_r be the cumulative number of position decisions after round r. The cosine schedule is

$$C_0 = 0, \qquad C_r = \max\Big\{ C_{r-1} + 1, \Big\lceil 272 \Big[1 - \cos\Big(\frac{\pi r}{128} \Big) \Big] \Big\rceil \Big\},$$

with the final cumulative count fixed at $C_{64} = 272$.

MaskGen uses classifier-free guidance scale $s = 1 + 2[32 - \max(1, M)]/32$, where M is the number of masked tokens.

For MAR-Sudoku with g clues, let $W = 81 - g$ denote the number of writable cells. A rethinking budget of R requires $D = W + 2R$ position decisions: $W + R$ fills and R removals. If round t makes K_t decisions and the trajectory has L rounds, then

$$\sum_{t=1}^{L} K_t = D.$$

All K_t decisions in one round share the same generator features, giving L backbone evaluations. For Sudoku, $K_t \in \{1, 2\}$. When $W \geq 2R + 1$, using $K_t = 2$ for $2R$ rounds and $K_t = 1$ for the remaining $W - 2R$ rounds gives $L = W$. Thus, on an empty grid with the training budget $R = 9$, the policy makes 99 position decisions over 81 backbone evaluations.

Table 7: Generator architectures and dynamic-removal inference settings.

Parameter	MAR-B	MaskGen-KL-XL	MAR-Sudoku
Architecture			
Tokenizer	KL16 VAE	TA-TiTok-BL32 VAE	MNIST VAE
Token layout	16×16 grid	32-token sequence	9×9 grid
Number of tokens	256	32	81
Token dimension	16	16	32
Backbone	encoder–decoder	MM-DiT decoder	encoder–decoder
Model width	768	1280	256
Encoder depth	12	—	3
Decoder/joint depth	12	20	3
Attention heads	12	16	4
Feed-forward width	3072	5120	1024
Generative head	DiffLoss	DiffLoss	FlowMatch
Head depth	6	8	3
Head width	1024	1280	256
Frozen parameters	207.925M	1237.345M	6.215M
Inference			
Sampler	DDIM ($\eta = 0$)	DDIM ($\eta = 0$)	Euler
Denoising steps	25	25	12
CFG schedule	constant 1.0	revealed-token linear	none
Parallelization	64-round cosine	sequential	2 actions/round, then 1
Selections per round	1–7	1	1–2
Rounds	64	40	81 (empty grid)
Removal budget	8	4	9
Position decisions	272	40	99 (empty grid)

Table 8: Test-time scaling of the PONDER policy trained with $R = 9$ on MNIST Sudoku. Overall and hard-split exact-solution accuracy are measured on 30,000 held-out boards balanced across difficulty levels.

R	Overall (%) ↑	Hard (%) ↑	Medium (%) ↑	Easy (%) ↑
0	80.17	56.60	87.07	96.85
1	89.68	75.50	94.22	99.33
2	93.62	84.90	96.50	99.46
4	96.53	92.52	97.53	99.53
8	98.07	96.39	98.30	99.52
9	98.21	96.68	98.39	**99.57**
16	98.56	97.72	98.60	99.37
32	98.68	98.07	98.60	99.36
40	98.72	98.13	98.57	99.47
64	98.75	98.27	**98.70**	99.27
128	**98.89**	**98.61**	98.64	99.41

MAR-B and MaskGen-KL-XL use released pretrained weights without fine-tuning. MAR-Sudoku is trained for 500,000 updates with AdamW, $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 0.2, and a cosine learning-rate schedule from 8×10^{-4} to 1×10^{-4}.

A.3 SUDOKU TEST-TIME SCALING

Table 8 reports accuracy across rethinking budgets.

At $R = 9$, accuracy is 98.21%, 98.02%, and 97.94% across three evaluation seeds of the same policy, compared with 80.17%, 79.96%, and 80.29% at $R = 0$. For the first seed, the improvement is 18.04 percentage points.

A.4 MAR-B TEST-TIME SCALING

Table 9 reports additional PONDER budgets together with random-policy baselines.

Table 9: Class-conditional ImageNet results with the frozen MAR-B generator. The FIDgrad policy is trained with a rethinking budget of $R = 8$ and evaluated at every listed test-time budget R. We generate 50,000 images (50 per class) with DDIM using 25 denoising steps and guidance scale 1.

Position policy	R	Decisions	FID ↓	FDr-5 ↓	IS ↑
Random	0	256	4.09	13.69	181.22
Random	8	272	3.97	13.42	182.22
Random	32	320	3.80	12.82	185.90
PONDER	0	256	2.78	13.28	206.55
PONDER	1	258	2.78	13.22	206.13
PONDER	2	260	2.78	13.18	207.06
PONDER	4	264	2.75	13.09	207.78
PONDER	8	272	2.74	12.96	209.48
PONDER	16	288	2.72	12.70	212.05
PONDER	32	320	2.70	12.47	212.60

FDr-5 evaluation. FDr-5 is the mean of the normalized Fréchet distances in ConvNeXt, DINOv2, MAE, SigLIP, and CLIP representations, excluding the Inception reward space, using the FD-Loss evaluator (Yang et al., 2026).

A.5 MASKGEN PREFERENCE TEST-TIME SCALING

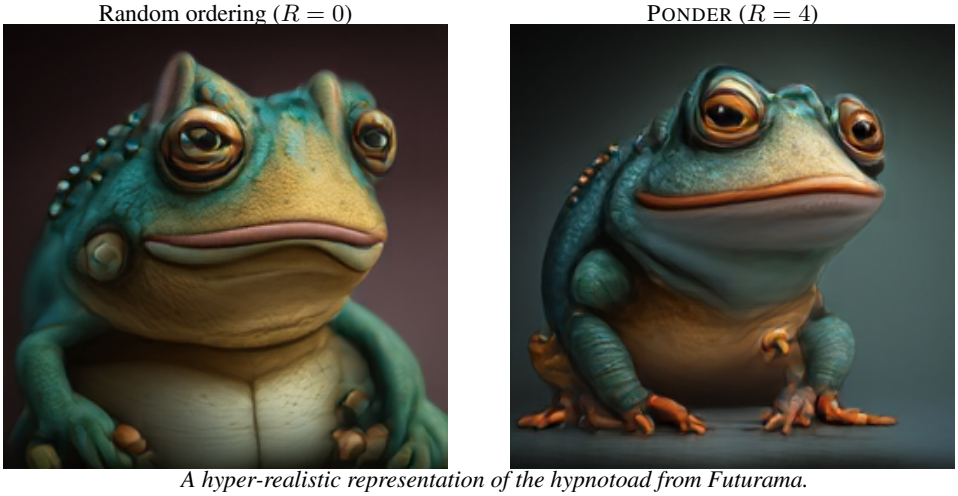
At test time, the policy uses one position decision per round, giving $32 + 2R$ rounds for $R \in \{0, 1, 2, 4, 8, 16, 32\}$.

Training uses the official HPSv2 training prompts for the reward; evaluation uses all 3,200 HPSv2 benchmark prompts. All learned-policy budgets use the same start noise. We report HPSv2, PickScore, and ImageReward.

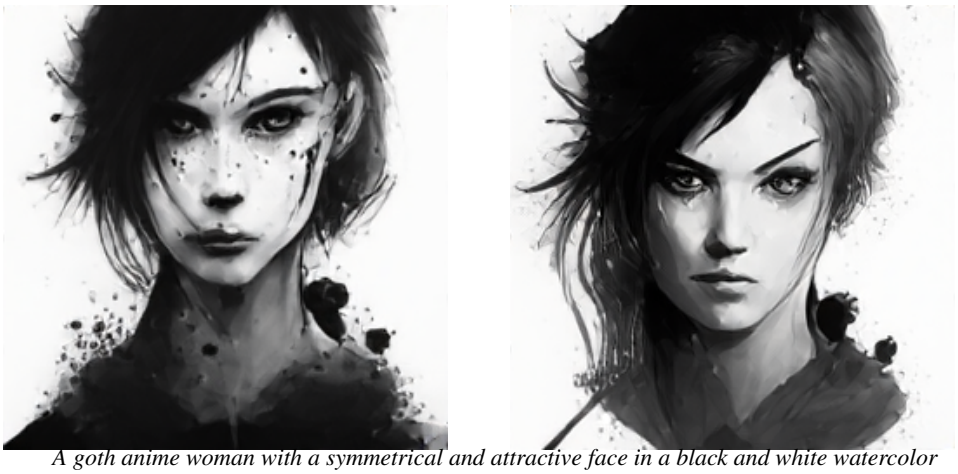
A.6 QUALITATIVE EXAMPLES

Figures 10 and 11 compare random-order and policy-guided MaskGen-KL-XL samples. Figure 12 shows MAR-B ImageNet samples, and Figure 13 illustrates patch removal and later regeneration during MAR-B sampling.

Random ordering ($R = 0$) | PONDER ($R = 4$)



A hyper-realistic representation of the hypnotoad from Futurama.



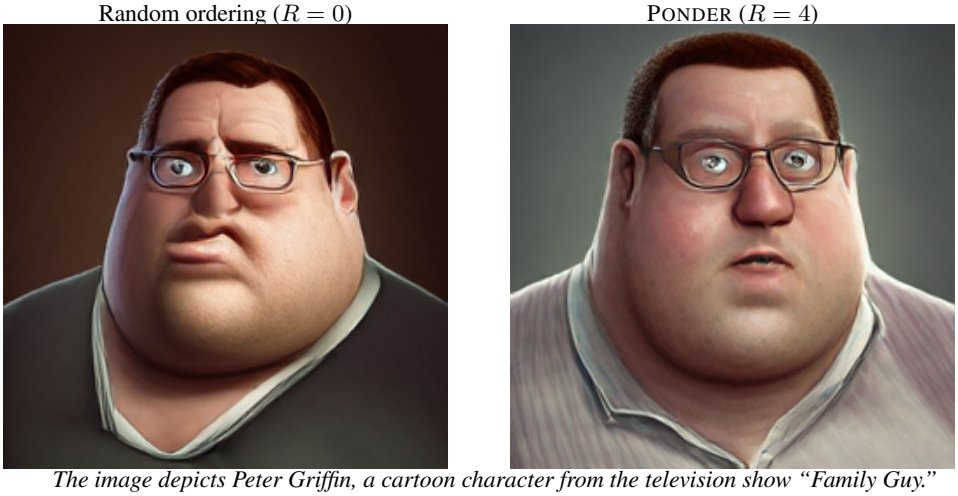
A goth anime woman with a symmetrical and attractive face in a black and white watercolor headshot art on ArtStation.



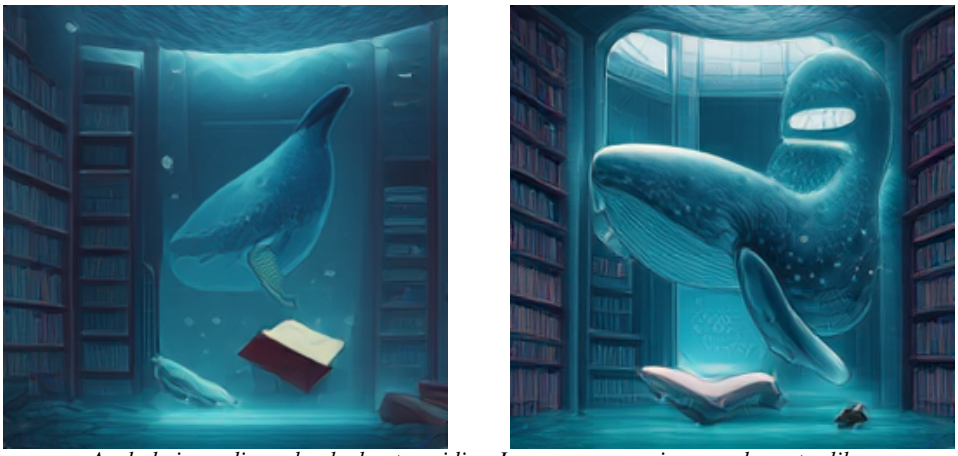
A Pixar lemon wearing sunglasses on a beach.

Figure 10: **MaskGen-KL-XL examples.** Random ordering at $R = 0$ (left) versus the HPSv2 policy at $R = 4$ (right), using the same prompts and DDIM-25 sampler.

Random ordering ($R = 0$) PONDER ($R = 4$)



The image depicts Peter Griffin, a cartoon character from the television show "Family Guy."



A whale is reading a book about avoiding Japanese spears in an underwater library.

Figure 11: **Additional MaskGen-KL-XL examples.** Random ordering at $R = 0$ (left) versus the HPSv2 policy at $R = 4$ (right), using the same prompts and DDIM-25 sampler.

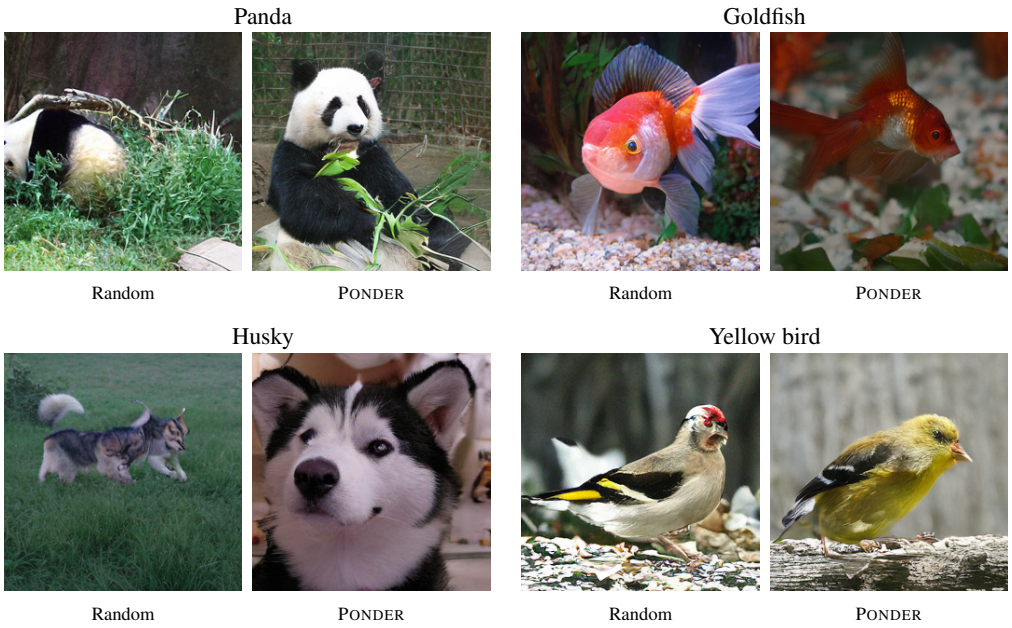


Figure 12: **MAR-B ImageNet qualitative examples.** Random-policy and PONDER samples with a rethinking budget of $R = 8$. Each class is shown as a random/PONDER pair.

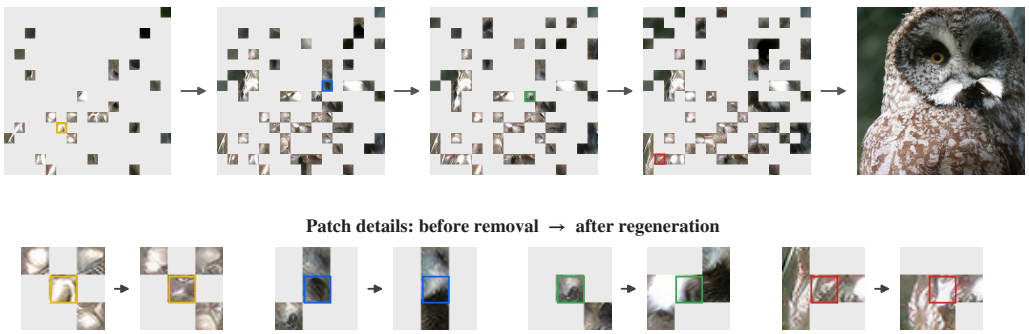


Figure 13: **MAR-B rethinking trajectory.** Each upper sequence shows successive partial canvases and the final image; the lower strip enlarges patches before removal and after regeneration. Colored boxes identify the revised locations.