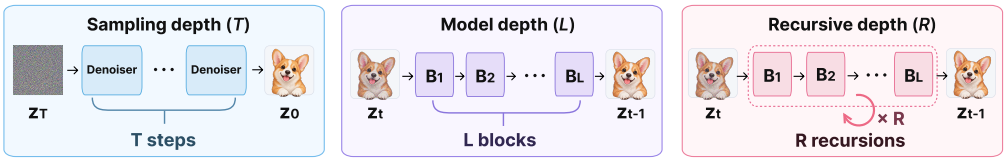
R-DiT: Recursive Diffusion Transformers

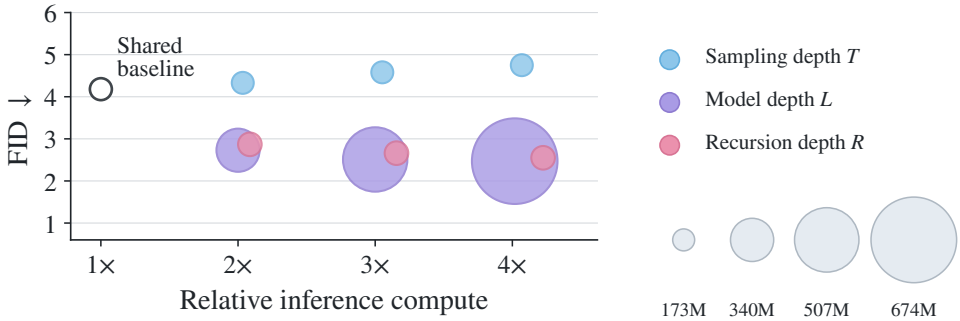
Anonymous authors
Paper under double-blind review

Abstract

Diffusion models have primarily scaled along two axes: increasing the number of model estimations and increasing the capacity of the denoiser. The former exhibits diminishing returns as sampling depth grows, while the latter improves generation at the cost of increasingly large parameter and memory footprints. We introduce **Recursive Diffusion Transformer (R-DiT)**, which introduces *recursive depth* as an additional compute-scaling axis. R-DiT repeatedly refines its hidden representation through shared Transformer blocks, increasing within-step computational depth without a proportional increase in the number of unique parameters. To make shared computation effective, we introduce **recursion modulation**, which conditions the denoiser jointly on diffusion time and recursion time. We further propose **time-adaptive recursion**, allocating different amounts of recursive computation across the denoising trajectory under a fixed compute budget. Across image, language, and molecular diffusion, we show that R-DiT significantly improves the quality–compute–parameter trade-off over standard diffusion Transformers. Our results establish recursive computation as a complementary scaling axis alongside sampling depth and model size in diffusion models.



(a) Three scaling axes for diffusion models.



(b) Scaling behavior of each axis.

Figure 1: **Recursive Diffusion Transformers (R-DiTs) introduces recursion depth as an additional scaling axis for diffusion transformers. (a)** Diffusion Computation can be scaled by increasing sampling depth T, model depth L, or recursion depth R. **(b)** Comparison of these three scaling strategies on class-conditional ImageNet 256×256 generation as inference computation increases. Starting from a shared baseline ($L = 7$, $T = 50$, $R = 1$), we scale each axis to approximately $2\times$, $3\times$, and $4\times$ the baseline compute. The x-axis shows measured inference FLOPs normalized by the baseline, including the small overhead introduced by recursion-specific modulation; marker diameter denotes model parameter count.

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.

1 INTRODUCTION

Diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020) have become a standard generative modeling paradigm across a wide range of domains. Their generation process is inherently iterative, progressively refining a sample through repeated evaluations of a learned denoiser. This provides two conventional ways to improve generation quality: performing more denoising steps during sampling, or strengthening the denoiser so that each step produces a more accurate update.

These two directions, however, offer different trade-offs. Increasing sampling depth can improve generation quality without increasing model size, but its gains eventually diminish (Karras et al., 2022; Ma et al., 2025). As illustrated in Figure 1b, additional evaluations bring little improvement beyond a certain point. Scaling the denoiser itself remains much more effective, and recent diffusion and flow models have achieved strong performance with increasingly large and deep Transformer backbones (Esser et al., 2024; Zhao et al., 2026; Ye et al., 2026). However, this improvement comes at the cost of a substantially larger parameter and memory footprint, making deployment on resource-constrained hardware increasingly costly (Li et al., 2023; Fang et al., 2025).

These complementary trade-offs motivate a natural question:

> 💡 Can we improve diffusion models without taking more denoising steps or building a larger denoiser?

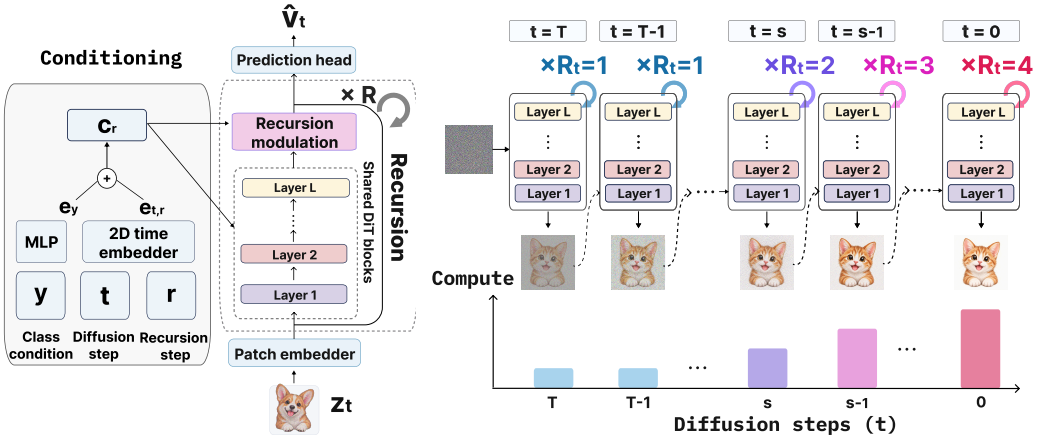
We explore recursion as a new scaling axis for diffusion. Conventional Diffusion Transformers (DiTs; Peebles and Xie 2023) increase within-step computational depth by stacking additional Transformer blocks, thereby coupling computational depth with parameter depth. Recursive or looped Transformers provide a natural alternative by repeatedly reusing the same blocks to increase effective computational depth without proportionally increasing the number of unique parameters (Dehghani et al., 2018; Giannou et al., 2023; Bae et al., 2026). Motivated by this perspective, we study *recursion depth* as an additional scaling axis for diffusion models, alongside sampling depth and model depth.

Applying recursion to diffusion, however, is not straightforward. Unlike standard recursive Transformers, recursion here unfolds inside an already time-conditioned denoising process, so the shared backbone must account for both diffusion progress and recursion progress. Moreover, the value of additional within-step computation need not be uniform across the denoising trajectory: different timesteps may benefit from different amounts of recursive refinement.

We introduce **Recursive Diffusion Transformer (R-DiT)**, a novel DiT architecture that recursively refines hidden representations within each denoising step using shared Transformer blocks. A naive design would apply the same transformation at every recursive pass because the backbone is shared, even though the hidden states serve different roles as refinement progresses. To address this, we introduce **recursion modulation**, which conditions the shared computation on both diffusion timestep and recursion step, allowing the same parameters to perform different refinement operations across passes. We further introduce **time-adaptive recursion**, which varies the number of recursive updates across diffusion timesteps, allocating computation where it is most useful under a fixed inference budget.

Empirically, R-DiT improves consistently as recursion depth increases across multiple model capacities. Under comparable inference budgets, recursive scaling recovers much of the benefit of model scaling while requiring far fewer unique parameters, and it provides larger gains than adding more sampling steps once sampling depth has saturated. For example, on ImageNet 256×256 (Deng et al., 2009), starting from a 7-layer DiT baseline, increasing sampling depth from $T = 50$ to $T = 200$ yields little improvement, whereas increasing recursion depth to $R = 4$ substantially improves FID and approaches the performance of a much deeper 28-layer denoiser, without a proportional increase in parameter count.

We also observe consistent benefits beyond image generation, suggesting that the effect of recursion depth scaling is not specific to a single diffusion modality. On language diffusion, a 3-layer backbone with recursion depth $R = 8$ reduces LM1B (Chelba et al., 2013) test PPL from 43.4 to 34.6. On molecular graph diffusion over MOSES (Polykovskiy et al., 2020), recursion increases molecular validity from 56.0% to 95.4%.



(a) R-DiT architecture (b) Time-adaptive recursion

Figure 2: **Recursive Diffusion Transformer (R-DiT). (a)** At each diffusion timestep t, R-DiT repeatedly refines an intermediate hidden representation using a shared Transformer backbone. Each recursive pass is conditioned jointly on diffusion time t, and recursion step r through recursion-aware modulation. **(b)** R-DiT generalizes a fixed recursion depth R to a time-dependent depth R_t, allowing within-step computation to be allocated non-uniformly along the denoising trajectory.

We summarize our contributions as follows:

- We introduce **R-DiT**, a recursive diffusion Transformer that adds *recursion depth* as a new scaling axis by repeatedly refining hidden representations within each denoising step using shared Transformer blocks.
- We propose **recursion-aware diffusion computation**, including **recursion modulation** to condition shared computation on diffusion timestep and recursion step, and **time-adaptive recursion** to allocate recursive computation non-uniformly across diffusion timesteps.
- We empirically show that recursive depth is a strong complementary scaling axis to sampling depth and model depth, improving generation quality across model capacities while requiring far fewer additional parameters than deeper denoisers.
- We further demonstrate the effectiveness of recursive computation beyond image generation, including language and molecular diffusion.

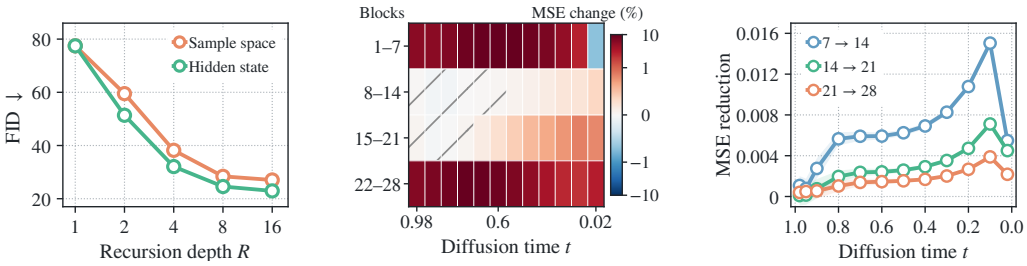
2 Background and Motivation

2.1 Diffusion and Flow Models

Diffusion and flow models generate samples by transporting random noise toward the data distribution through a learned time-dependent denoising or velocity field (Song et al., 2020; Ma et al., 2024). Modern formulations commonly parameterize this field with Diffusion Transformers (DiTs; Peebles and Xie 2023), which apply a Transformer-based denoiser repeatedly along the sampling trajectory and have scaled effectively (Nie et al., 2026; Ye et al., 2026). Further backgrounds in Appendix B.1.

2.2 Recursive Transformers

Recursive Transformers, also referred to as looped or recurrent Transformers, reuse the same Transformer layers across multiple computational passes. Given an initial representation $\mathbf{h}^{(0)}$, a shared transformation F_θ is repeatedly applied as $\mathbf{h}^{(r+1)} = F_\theta(\mathbf{h}^{(r)}; r)$, where $r = 0, \ldots, R-1$ indexes the recursion step and R denotes the recursion depth. For a backbone with L unique layers, recursion increases the effective computational depth from L to approximately $L \cdot R$ while retaining only L parameterized layers, thereby decoupling computational depth from parameter depth.



(a) Recursion space (b) Stage sensitivity (c) Marginal depth benefit

Figure 3: **Motivating observations. (a)** Hidden-state recursion achieves lower FID than token-space recursion for every R. **(b)** Sensitivity to stage attenuation varies with stage position and diffusion time. **(c)** Additional depth yields larger denoising-error reductions during late denoising.

2.3 Why Recursion for Diffusion?

Diffusion models have two distinct notions of computational depth: *sampling depth*, which determines how many times the generative state is updated, and *within-step depth*, which determines how much computation is used to produce each update.

Recursion provides a complementary compute axis. Increasing sampling depth T adds inference computation without increasing model size, but its benefits eventually diminish (Ho et al., 2020; Ma et al., 2025). Increasing model depth L, in contrast, strengthens within-step computation but also increases the number of unique parameters. Recursion provides another possibility to increase within-step computational depth without proportionally increasing the parameter size. Repeatedly applying a shared backbone within a fixed diffusion timestep increases within-step computational depth while largely reusing the same parameters. We therefore study recursion depth R as an additional compute-scaling axis alongside sampling depth T and model depth L.

Recursion is more effective in hidden space. Recursive computation can either repeatedly return to the diffusion-token representation (token-space recursion) or continue directly over the intermediate Transformer representation (hidden-state recursion). As shown in Figure 3a, both approaches benefit from increasing recursion depth, but hidden-state recursion consistently performs better over all recursion depths. This motivates preserving the evolving hidden representation across recursive passes and delaying the output projection until the final pass.

The value of depth depends on diffusion time. We analyze single-pass ($R = 1$) DiTs using held-out denoising MSE. We divide a 28-layer model into four seven-layer stages, attenuate one stage's hidden update at a time, and measure the resulting change in final prediction error. Figure 3b shows each group has distinct patterns over the diffusion time: for example, attenuating blocks 15–21 has little effect early but becomes more harmful during late denoising. We next compare independently trained models of increasing depth and compare validation denoising MSE over different timesteps. As shown in Figure 3c, additional depth yields larger MSE reductions during late denoising, before the gains decrease near the endpoint. These stage- and time-dependent effects motivate conditioning shared refinement on recursion progress and diffusion time, and allocating recursion through a time-dependent budget R_t, which we introduce in the next section. Details of the analysis are provided in Appendix D.1.

3 Recursive Diffusion Transformer

3.1 Two-Dimensional Time Conditioning

Standard Diffusion Transformers reuse the same denoising network across diffusion timesteps, while employing distinct parameters across network depth. Timestep conditioning modulates the network's activations according to the current noise level. R-DiT additionally shares backbone parameters across repeated passes within each denoising evaluation. We therefore introduce the discrete

recursion index r as a second computational coordinate and condition the model jointly on diffusion time t, recursion index r, and class label y.

Specifically, starting from $\mathbf{h}_t^{(0)} = \mathbf{E}(z_t)$, we write

$$\mathbf{h}_t^{(r+1)} = F_\theta\Big(\mathbf{h}_t^{(r)}; t, r, y\Big), \qquad r = 0, \ldots, R_t - 1, \tag{1}$$

where y denotes class conditioning and θ is shared across passes. Diffusion time t remains fixed during recursion, while r identifies the stage of within-step refinement.

A conditioned prediction head reads only the final hidden state:

$$\hat{\mathbf{v}}_t = \mathbf{g}\Big(\mathbf{h}_t^{(R_t)}; t, R_t - 1, y\Big). \tag{2}$$

Thus, R_t controls computation within a denoising step without adding unique backbone layers or additional denoising transitions.

3.2 Recursive Modulation

To control how much the hidden representation changes across passes of a shared backbone, we preserve the previous state through a residual path and learn a conditioned refinement.

Let B_θ denote the shared backbone and $\mathbf{c}_{t,r}$ the joint conditioning on diffusion time, recursion step, and class type. The condition predicts $(\beta_{t,r}, \gamma_{t,r})$; $\mathbf{p}_\psi$ is a conditioned update controller, and $\alpha_{t,r}$ is a learned scalar gate. The first pass initializes $\mathbf{h}_t^{(1)} = \mathbf{b}_\theta(\mathbf{h}_t^{(0)}; c_{t,0})$. Subsequent passes apply

$$\tilde{\mathbf{h}}_t^{(r)} = (1 + \gamma_{t,r}) \odot \mathrm{LN}(\mathbf{h}_t^{(r)}) + \beta_{t,r}, \tag{3}$$

$$\mathbf{h}_t^{(r+1)} = \mathbf{h}_t^{(r)} + \alpha_{t,r}\mathbf{p}_\psi\Big(\mathbf{b}_\theta(\tilde{\mathbf{h}}_t^{(r)}; c_{t,r}); c_{t,r}\Big), \tag{4}$$

These recursive updates separate the persistent representation from its pass-specific refinement while reusing the same backbone parameters. Architectural and initialization details are provided in the Appendix.

3.3 Time-Adaptive Recursion

Our analysis in Sec. 2.3 shows that later denoising steps benefit most from deeper models. Throughout experiments, we simply choose a linear schedule, increasing the target recursion depth from $R_{\min} = \bar{R}/2$ to $R_{\max} = 3\bar{R}/2$ over the sampling-step index. We round these targets to integer passes while preserving the average depth $\bar{R}$, allocating more computation to later steps without increasing the total recursion budget.

The schedule changes only the number of shared-backbone passes within each denoising step while the sampler and its timesteps remain unchanged. During training, we draw timesteps independently and assign each example its corresponding depth. Schedule construction and training details are provided in the Appendix D.3.

Comparison with ELT Closest to our work, ELT (Goyal et al., 2026) applies recursive mechanism in image generation. However, its recursion operates directly at the token-space level and uses a fixed amount of recursive computation, without considering how the required computation may vary across diffusion timesteps. In our experiments, such uniform allocation is suboptimal. In contrast, R-DiT performs recursion over hidden representations within each denoising step, using recursion-aware modulation to differentiate successive passes and adaptive recursion to allocate computation across diffusion time. Together, these designs enable recursive computation to transfer effectively to diffusion models.

4 Experiments

We organize our experiments around four questions: (i) how does recursive depth scale across model capacities and relative to sampling and model depth? (ii) how important are recursion-aware conditioning and modulation? (iii) how should recursive computation be allocated across diffusion time? and (iv) do these findings extend beyond image generation?

Table 1: **Comparison of three diffusion scaling axes for class-conditional ImageNet 256×256 generation.** Starting from the same baseline, we increase inference computation through sampling depth T, model depth L, or average recursion depth $\bar{R}$.

Scaling axis	T	L	$\bar{R}$	#Params	TFLOPs	FID ↓	IS ↑	sFID ↓	Precision ↑	Recall ↑
Baseline	50	7	1	173M	5.8	4.18	212.5	6.32	0.746	0.558
Sampling depth (T)	100	7	1	173M	11.8	4.33	213.7	6.29	0.745	0.556
	150	7	1	173M	17.7	4.58	210.8	6.62	0.739	0.555
	200	7	1	173M	23.6	4.75	209.1	6.86	0.735	0.558
	250	7	1	173M	29.6	4.82	209.2	7.02	0.733	0.554
Model depth (L)	50	14	1	340M	11.6	2.73	243.1	5.49	0.793	0.573
	50	21	1	507M	17.4	2.51	**251.6**	5.32	**0.807**	0.572
	50	28	1	674M	23.3	**2.47**	247.8	5.46	0.806	0.578
Recursion depth ($\bar{R}$)	50	7	2	186M	12.1	2.87	241.2	5.16	0.771	0.578
	50	7	3	186M	18.3	2.66	236.9	**5.03**	0.787	**0.582**
	50	7	4	186M	24.5	2.55	243.7	5.07	0.794	0.577

4.1 Image Generation Setup

Image generation. We conduct our main image-generation experiments on class-conditional ImageNet-256×256 (Deng et al., 2009) under two complementary settings. For controlled studies, we follow the original DiT setup and use a compact backbone with $L = 3$ unique Transformer layers and hidden dimension $d = 768$ unless otherwise specified. We use this setting to study recursion scaling and architectural design choices.

We additionally evaluate R-DiT under the stronger VA-VAE-based diffusion setup (Yao et al., 2025). Unless otherwise stated, this setting uses hidden dimension $d = 1152$ and serves as our primary setup for comparing sampling depth T, model depth L, and recursion depth R, as well as for adaptive and larger-scale experiments. Within each comparison, all other training and sampling components are held fixed. Additional implementation details are provided in Appendix C.3.

Evaluation. We follow the standard ImageNet evaluation protocol and compute all generation metrics from 50K samples. We use Fréchet Inception Distance (FID; Heusel et al. 2017) as our primary metric and additionally report Inception Score (IS; Salimans et al. 2016), spatial FID (sFID; Nash et al. 2021), Precision, and Recall (Kynkäänniemi et al., 2019). For scaling experiments, we also report model parameter count and measured inference FLOPs.

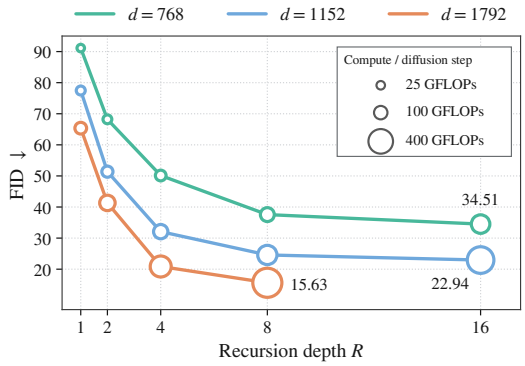
4.2 Recursion Depth as an Additional Scaling Axis

Recursion scaling across model capacities. We first isolate the effect of recursion depth in the controlled DiT setting. Keeping the number of unique layers fixed at $L = 3$, we vary R across backbones with hidden dimensions $d \in \{768, 1152, 1792\}$. As shown in Figure 4, increasing R consistently improves generation quality across all three model capacities. At $R = 8$, FID decreases from 91.12 to 37.6, from 77.5 to 24.6, and from 65.4 to 15.6 for $d = 768$, 1152, and 1792, respectively.

The gains persist as the underlying denoiser becomes stronger, indicating that recursion is complementary to model capacity rather than merely compensating for a small backbone. Interestingly, the useful range of recursion depth also appears to increase with model capacity: the $d = 768$ model improves through $R = 16$ before saturating, while the $d = 1152$ model continues to improve through $R = 16$ and the $d = 1792$ model through the largest depth tested. This suggests that stronger backbones can make effective use of deeper recursive computation.

Comparison with sampling and model scaling. We next compare recursion depth with the two conventional ways of increasing diffusion computation: sampling depth T and model depth L. In the VA-VAE setting, we scale each compute axis from the same shared baseline.

The three strategies exhibit markedly different scaling behavior. Increasing sampling depth does not improve the CFG-tuned baseline: FID changes from 4.18 at $T = 50$ to 4.33 at $T = 100$ and



(a) Recursive scaling across model capacities

Method	Epochs	R	FID↓	IS↑	sFID↓	Pre.↑	Rec.↑
DiT-XL/2 [27]	1400	–	2.27	278.2	4.60	0.83	0.57
SiT-XL/2 [22]	1400	–	2.06	270.3	4.50	0.82	0.59
REPA [38]	800	–	1.42	305.7	4.70	0.80	0.65
LightningDiT [36]	800	–	1.35	295.3	4.15	0.79	0.65
		1	2.08	253.1	4.29	0.81	0.59
R-DiT	**64**	2	1.92	254.4	4.48	0.79	**0.62**
		3	**1.79**	**281.2**	**3.95**	**0.81**	0.61

(b) Recursion on the XL-baseline

Figure 4: **Recursive scaling remains effective at larger model capacities. (a)** Increasing recursion depth improves generation quality across backbones with $d \in \{768, 1152, 1792\}$ while keeping the number of unique Transformer layers fixed. **(b)** Recursion scaling with large-backbone model XL (28-layers) setup; our model is trained for only 64 epochs.

gradually degrades as T increases further. Increasing model depth remains highly effective, reaching FID 2.47 at $L = 28$, but increases the parameter count from 173M to 674M. Recursion depth scaling reaches FID 2.55 at approximately the $4\times$ compute budget with only 186M parameters, while retaining the same 7 unique Transformer layers.

These results reveal a distinct quality–compute–parameter trade-off: recursive depth increases within-step computational depth while largely reusing the same parameterized backbone.

Scaling to larger backbones. Finally, we evaluate recursion on an already deep, 28-layer VA-VAE backbone. At $T = 250$, increasing recursion depth from $R = 1$ to $R = 2$ and $R = 3$ reduces FID from 2.08 to 1.92 and 1.79, respectively. At $R = 3$, the model additionally reaches IS 281.2 and sFID 3.95. As summarized in Figure 4(b), this performance is obtained after only 64 training epochs. These results show that recursive computation remains beneficial even when applied to a substantially stronger and deeper denoiser.

4.3 Recursion-Aware Design

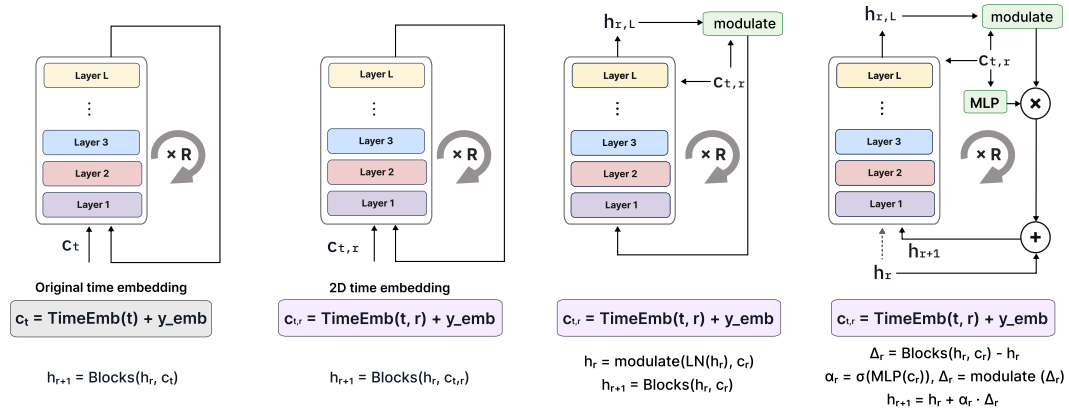
Because the same Transformer blocks are reused across recursive passes, they must operate on hidden states at different stages of refinement. We therefore ablate how the shared backbone is conditioned on the two coordinates of R-DiT: diffusion time t and recursion step r. Figure 5 compares (a) standard diffusion-time conditioning, (b) a *2D time embedding* over (t, r), (c) recursion-aware adaLN modulation, and (d) the full R-DiT design with gated residual updates.

While all variants benefit from increasing recursion depth, recursion progress through the 2D time embedding provides a substantial additional gain. Recursion-aware strength modulation further allows the shared layers to perform different transformations at different (t, r) coordinates. Finally, the gated residual update allows each recursive pass to control the magnitude of its hidden-state refinement, rather than applying every update with the same strength.

These components provide complementary improvements. For example, at $R = 8$ and CFG $= 1.5$, FID improves from 22.12 with the original conditioning to 19.24 with the 2D time embedding, 18.97 with recursion-aware modulation, and 17.01 with the full gated design. The same ordering is observed under CFG $= 1.0$. Together, these results support a recursion-aware design in which (t, r) specifies both *how* the shared backbone should transform its representation and *how strongly* each recursive pass should update it. Full metrics and additional results are provided in Appendix D.2.

4.4 Understanding Adaptive Recursion

We next examine how recursive computation should be distributed across the denoising trajectory. Using the controlled DiT setting, we partition diffusion time into four regions from high to low

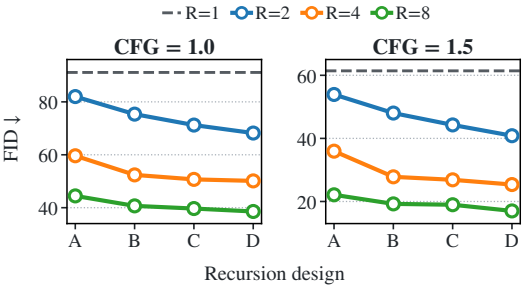


(a) Naive sharing (b) 2D embedding (c) Strength modulation (d) Gated residual

Figure 5: **Design choices of recursion architecture.** We progressively augment the original diffusion-time conditioning with joint time–recursion conditioning, recursion-aware modulation, and gated residual updates.

Table 2: **(Left) Adaptive allocation of recursive computation across diffusion time.** A schedule (R_1, R_2, R_3, R_4) specifies the recursion depth assigned to four diffusion-time regions ordered from high to low noise; for example, $(6, 6, 6, 6)$ applies six recursive passes uniformly across all regions. All schedules use the same recursive-compute budget. **(Right) Recursion-aware design ablations.** Increasing recursion depth improves generation quality across all variants, while recursion-aware conditioning and modulation provide additional gains at the same recursion depth under both CFG settings.

Schedule	FID ↓	IS ↑	sFID ↓	Pre. ↑	Rec. ↑
Early-heavy					
8, 6, 6, 4	17.9	77.6	5.45	0.68	**0.54**
Uniform					
6, 6, 6, 6	15.8	84.9	5.25	0.70	0.53
Late-heavy					
4, 6, 6, 8	15.1	87.6	5.29	0.71	0.53
2, 6, 8, 8	14.4	90.1	5.18	0.71	0.53
1, 7, 8, 8	**14.0**	**92.7**	**5.16**	**0.72**	0.52



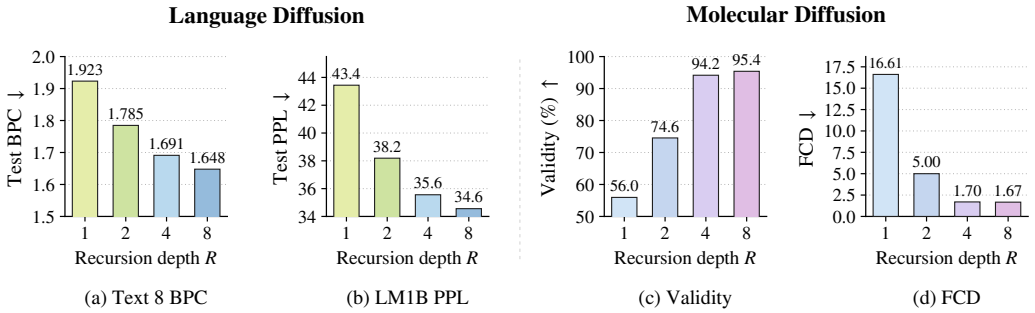
noise and vary the recursion depth assigned to each region while keeping the total recursive-compute budget approximately fixed.

Recursive compute is more valuable at later timesteps. As shown in Table 2 (left), reallocating a fixed recursion budget toward later, lower-noise stages consistently improves FID and IS over uniform recursion. The late-heavy schedule $(1, 7, 8, 8)$ improves FID from 15.8 to 14.0 and IS from 84.9 to 92.7, whereas the early-heavy schedule $(8, 6, 6, 4)$ degrades performance. These results indicate that the marginal value of recursive computation varies across diffusion time, with later denoising stages benefiting more from additional refinement. The same trend is observed under CFG $= 1.0$; further results are provided in Appendix D.3.

4.5 Generalization Beyond Image Generation

Finally, we study whether recursive computation is specific to image diffusion or represents a more general architectural principle. For each domain, we compare the original diffusion backbone with its recursive counterpart under comparable parameter and compute budgets.

Language diffusion. We apply R-DiT to masked diffusion language modeling following MDLM (Sahoo et al., 2024), using a 3-layer Transformer with $R \in \{1, 2, 4, 8\}$ refinement passes. At equal training-update budgets, increasing R from 1 to 8 reduces Text8 test BPC from 1.923 to 1.648 and LM1B test perplexity from 43.4 to 34.6, using EMA weights and ELBO-based evaluation.



(a) Text 8 BPC (b) LM1B PPL (c) Validity (d) FCD

Figure 6: **Recursive scaling generalizes beyond image generation. (a–b)** In language diffusion, increasing R improves test BPC on text 8 and test PPL for LM1B. **(c–d)** In molecular diffusion, increasing R improves molecular validity and distribution fidelity measured by FCD.

These gains require only a modest increase in unique parameters. Full configurations and additional results are provided in Appendix D.4.

Molecular graph diffusion. We further evaluate R-DiT on MOSES (Polykovskiy et al., 2020) using MolHIT (Jung et al., 2026) as the base model. We use a 3-layer graph diffusion Transformer (Vignac et al., 2022), train models with $R \in \{1, 2, 4, 8\}$ for 50 epochs, and evaluate 10K generated molecules with 500 sampling steps. Recursive scaling substantially improves molecular generation quality. Increasing R from 1 to 8 raises validity from 56.0% to 95.4% and reduces FCD (Preuer et al., 2018) from 16.61 to 1.67. We additionally find that adaptive recursion further improves performance, and that the benefits of recursion remain consistent across different diffusion samplers. Additional results are provided in Appendix D.5.

5 Related Works

Diffusion model architectures Early diffusion models predominantly relied on convolutional U-Net denoisers (Ho et al., 2020; Dhariwal and Nichol, 2021). Diffusion Transformers (DiTs) (Peebles and Xie, 2023) replaced this backbone with Transformer blocks and demonstrated strong scaling with model capacity. Subsequent work has further improved Transformer-based diffusion through better training objectives, representation learning, and latent/tokenizer design (Ma et al., 2024; Yu et al., 2024; Yao et al., 2025). These approaches primarily improve within-step computation by strengthening or enlarging the denoiser, typically coupling additional computational depth with additional parameterized layers. R-DiT instead increases within-step depth by reusing a shared backbone, decoupling computational depth from the number of unique Transformer layers.

Recursive Transformers. Recursive Transformers reuse shared blocks across depth to increase effective computational depth without a proportional increase in unique parameters (Dehghani et al., 2018; Lan et al., 2019; Giannou et al., 2023; Yang et al., 2024; Bae et al., 2025). More recent work treats recursion as an explicit test-time compute axis: recurrent-depth language models vary the number of latent refinement passes (Geiping et al., 2026), while Mixture-of-Recursions (MoR; Bae et al. 2026) allocates different recursion depths across tokens. R-DiT builds on this perspective in diffusion, where recursive computation is performed within each denoising step and jointly conditioned on diffusion time and recursion progress.

6 Conclusion

In this work, we present Recursive Diffusion Transformers (R-DiTs), which utilize recursion depth as another scaling axis for diffusion models. To make scaling depth effective, we introduce a recursion module based on 2D time embedding and adaptive recursion that allocates different compute for different denoising steps. Our experiments show R-DiT significantly outperforms the same parameter size DiTs in the equal-compute setup across image, language, and molecules.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish the writing of this paper and to search for and identify related work. We have reviewed the AI-assisted text and the retrieved references. We take responsibility for the final content of this work.

REFERENCES

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. In *International Conference on Learning Representations*, volume 2025, pages 34282–34327, 2025.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *Advances in Neural Information Processing Systems*, 38:96572–96617, 2026.

G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90–98, 2012.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1(1):8, 2009.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

Gongfan Fang, Kunjun Li, Xinyin Ma, and Xinchao Wang. Tinyfusion: Diffusion transformers learned shallow. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18144–18154. IEEE, 2025.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2026.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pages 11398–11442. PMLR, 2023.

Sahil Goyal, Swayam Agrawal, Gautham Govind Anil, Prateek Jain, Sujoy Paul, and Aditya Kusupati. Elt: Elastic looped transformers for visual generation. In *European Conference on Computer Vision*, pages 308–326. Springer, 2026.

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Hojung Jung, Rodrigo Hormazabal, Jaehyeong Jo, Youngrok Park, Kyunggeun Roh, Se-Young Yun, Sehui Han, and Dae-Woong Jeong. Molhit: Advancing molecular-graph generation with hierarchical discrete diffusion models. *arXiv preprint arXiv:2602.17602*, 2026.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35:26565–26577, 2022.

Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *Advances in neural information processing systems*, 32, 2019.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Seul Lee, Karsten Kreis, Srimukh Prasad Veccham, Meng Liu, Danny Reidenbach, Yuxing Peng, Saee Paliwal, Weili Nie, and Arash Vahdat. Genmol: A drug discovery generalist with discrete diffusion. *arXiv preprint arXiv:2501.06158*, 2025.

Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. *Advances in Neural Information Processing Systems*, 36:20662–20678, 2023.

Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In *European Conference on Computer Vision*, pages 23–40. Springer, 2024.

Nanye Ma, Shangyuan Tong, Haolin Jia, Hexiang Hu, Yu-Chuan Su, Mingda Zhang, Xuan Yang, Yandong Li, Tommi Jaakkola, Xuhui Jia, et al. Inference-time scaling for diffusion models beyond scaling denoising steps. *arXiv preprint arXiv:2501.09732*, 2025.

Matt Mahoney. Text8 dataset. http://mattmahoney.net/dc/textdata.html, 2011. Accessed: 2021-05-24.

Charlie Nash, Jacob Menick, Sander Dieleman, and Peter W Battaglia. Generating images with sparse representations. *arXiv preprint arXiv:2103.03841*, 2021.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.

Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in pharmacology*, 11:565644, 2020.

Kristina Preuer, Philipp Renz, Thomas Unterthiner, Sepp Hochreiter, and Gunter Klambauer. Fréchet chemnet distance: a metric for generative models for molecules in drug discovery. *Journal of chemical information and modeling*, 58(9):1736–1741, 2018.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr, 2015.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International conference on learning representations*, volume 2024, pages 42195–42214, 2024.

Jingfeng Yao, Bin Yang, and Xinggang Wang. Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15703–15712. IEEE, 2025.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think. *arXiv preprint arXiv:2410.06940*, 2024.

Bing Zhao, Chenfei Wu, Deqing Li, Hao Meng, Jiahao Li, Jie Zhang, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kuan Cao, et al. Qwen-image-2.0 technical report. *arXiv preprint arXiv:2605.10730*, 2026.

TABLE OF CONTENTS

1 Introduction **2**

2 Background and Motivation **3**

2.1 Diffusion and Flow Models 3

2.2 Recursive Transformers 3

2.3 Why Recursion for Diffusion? 4

3 Recursive Diffusion Transformer **4**

3.1 Two-Dimensional Time Conditioning 4

3.2 Recursive Modulation 5

3.3 Time-Adaptive Recursion 5

4 Experiments **5**

4.1 Image generation setup 6

4.2 Recursion depth as an additional scaling Axis 6

4.3 Recursion-Aware Design 7

4.4 Understanding Adaptive Recursion 7

4.5 Generalization Beyond Image Generation 8

5 Related works **9**

6 Conclusion **9**

A Broader Impact **15**

B Further background **15**

B.1 Additional Background on Diffusion and Flow Models 15

C Recursive Diffusion Transformer Details **16**

C.1 Architecture 16

C.2 Initialization 17

C.3 Image Generation Setup 17

C.3.1 Controlled DiT Setup with SD-VAE 17

C.3.2 VA-VAE Setup 17

D Experimental Details **18**

D.1 Analysis of Depth and Diffusion Time 18

D.2 Recursion-Aware Design Ablations 19

D.3 Time-Adaptive Recursion 20

D.4 Language Diffusion 20

D.5 Molecular Diffusion 21

A Broader Impact

Impact statement. R-DiT provides a way to increase the computational depth of diffusion models while reusing the same parameterized backbone. This may enable stronger generative models under constrained parameter and memory budgets, potentially making diffusion scaling more accessible on limited hardware. At the same time, R-DiT inherits the broader risks of the generative models to which it is applied, including the generation of misleading or harmful content. In scientific domains such as molecular generation, improved generative capability could also be misused to propose harmful compounds. Our method does not introduce mechanisms specifically intended to mitigate these downstream risks.

Limitations. Our study focuses on establishing recursion as an additional compute-scaling axis for diffusion models and characterizing its behavior across image, language, and molecular generation. While our experiments cover multiple model capacities and recursion depths, a broader characterization at substantially larger model and training scales remains an important direction. Recursion trades additional sequential computation for parameter reuse, so its wall-clock efficiency can depend on implementation and hardware even when parameter growth is small. Finally, our adaptive formulation varies recursion primarily with diffusion time, leaving more fine-grained, input- or state-dependent allocation of recursive computation unexplored.

Future directions. A natural next step is to scale recursive computation to substantially larger generative models, particularly modern text-to-image (Esser et al., 2024) and video/world models. These settings provide an important testbed for whether increasing computational depth through parameter reuse can remain effective as denoiser capacity and generation complexity grow. Another promising direction is to learn recursion depth dynamically from the timestep, input, or intermediate hidden state, and to jointly allocate compute across sampling depth, model depth, and recursion depth. We believe studying these interactions at scale may reveal more efficient ways to increase test-time computation in large generative models.

B Further Background

B.1 Additional Background on Diffusion and Flow Models

Forward diffusion process. Diffusion models define a time-dependent stochastic process that gradually transforms data samples $\mathbf{x}_0 \sim q$ into a simple reference distribution, typically Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(0, I)$. In continuous time, the forward process can be written as the stochastic differential equation (SDE)

$$d\mathbf{x}_t = f(\mathbf{x}_t, t)\, dt + g(t)\, d\mathbf{w}_t, \qquad t \in [0, T], \tag{5}$$

where f denotes the drift, g the diffusion coefficient, and $\mathbf{w}_t$ a standard Wiener process. For commonly used diffusion processes, the corresponding conditional marginal can be expressed in the Gaussian perturbation form

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I), \tag{6}$$

where α_t and σ_t determine the signal and noise levels along the trajectory.

Reverse-time generation. The forward process admits a corresponding reverse-time SDE whose drift depends on the time-dependent score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ (Song et al., 2020). When integrated from noise back toward the data distribution, the reverse process can be written as

$$d\mathbf{x}_t = \left[\mathbf{f}(\mathbf{x}_t, t) - g(t)^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\right] dt + g(t)\, d\bar{\mathbf{w}}_t, \tag{7}$$

where $\bar{\mathbf{w}}_t$ denotes the reverse-time Wiener process. In practice, a neural network is trained to estimate the score or an equivalent quantity, and generation is performed by numerically integrating the reverse dynamics over a finite set of timesteps.

The same marginal distributions can also be generated deterministically through the probability-flow ODE

$$\frac{d\mathbf{x}_t}{dt} = \mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t), \tag{8}$$

which replaces the stochastic reverse process with a deterministic vector field while preserving the same time-marginal distributions.

Prediction parameterizations. Rather than predicting the score directly, modern diffusion models often parameterize an equivalent target such as the injected noise, the clean sample, or a velocity field. Under the Gaussian interpolation in Eq. 6, the instantaneous velocity associated with the path is

$$\mathbf{v}_t = \dot{\alpha}_t \mathbf{x}_0 + \dot{\sigma}_t \epsilon. \tag{9}$$

A velocity-prediction model $v_\theta(\mathbf{x}_t, t)$ can therefore be trained with the regression objective

$$\mathcal{L}_{\text{vel}} = \mathbb{E}_{\mathbf{x}_0,\epsilon,t}\left[w(t)\left\|\mathbf{v}_\theta(\mathbf{x}_t,t) - \mathbf{v}_t\right\|_2^2\right], \tag{10}$$

where $w(t)$ denotes an optional time-dependent weighting. Score-, noise-, and velocity-prediction parameterizations are related by deterministic transformations for a fixed noise schedule.

Flow-based generative models. Flow matching provides a closely related deterministic formulation in which a time-dependent vector field is learned directly to transport samples between a reference distribution and the data distribution. Given a probability path p_t with associated conditional velocity $\mathbf{u}_t$, a model v_θ is commonly trained through

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t,\mathbf{x}_t}\left[\left\|\mathbf{v}_\theta(\mathbf{x}_t,t) - \mathbf{u}_t\right\|_2^2\right]. \tag{11}$$

Sampling then follows the learned ODE

$$\frac{d\mathbf{x}_t}{dt} = \mathbf{v}_\theta(\mathbf{x}_t, t). \tag{12}$$

Despite differences in training objective and path construction, diffusion and flow models therefore share the same central computational structure for our purposes: generation proceeds through repeated evaluations of a time-conditioned neural vector field.

Transformer denoisers. Diffusion Transformers (DiTs ; Peebles and Xie 2023) replace the convolutional denoiser traditionally used in diffusion models with a Transformer operating over image or latent-space tokens. For a backbone with L Transformer layers, each function evaluation performs L layers of within-step computation before producing a denoising or velocity prediction. Consequently, inference computation can be increased in at least two conventional ways: by increasing the number of function evaluations along the sampling trajectory or by increasing the number of Transformer layers used in each evaluation. R-DiT introduces recursion depth as an additional means of increasing the latter while reusing the same parameterized backbone.

C Recursive Diffusion Transformer Details

C.1 Architecture

Shared backbone. R-DiT contains L distinct Transformer blocks that are reused across R recursive passes within each denoising evaluation. Parameters are shared across recursion passes, but not across the L blocks. The noisy latent is patch-embedded once before recursion and augmented with fixed two-dimensional sinusoidal positional embeddings.

Diffusion and recursion conditioning. Each recursive pass is conditioned on diffusion time t, recursion step r, and class label y. We form a two-dimensional time embedding as

$$\mathbf{c}_{t,r} = \text{MLP}\left([\text{FF}(t), \text{FF}(r)]\right) + \mathbf{e}_y, \tag{13}$$

where $\text{FF}(\cdot)$ denotes the sinusoidal feature embedding and $\mathbf{e}_y$ is the class embedding. The same conditioning vector is used by the Transformer modulation, recurrent update, and prediction head.

Hidden-state recursion. The first pass initializes the hidden representation as

$$\mathbf{h}_t^{(0)} = F_\theta\left(E(\mathbf{z}_t) + \mathbf{p}; \mathbf{c}_{t,0}\right), \tag{14}$$

where E is the patch embedding and $\mathbf{p}$ is the positional encoding. Subsequent passes update this representation as

$$\mathbf{u}_t^{(r)} = F_\theta\left(A_\theta(\text{LN}(\mathbf{h}_t^{(r-1)}), \mathbf{c}_{t,r}); \mathbf{c}_{t,r}\right), \tag{15}$$

$$\mathbf{h}_t^{(r)} = \mathbf{h}_t^{(r-1)} + \sigma(g_\theta(\mathbf{c}_{t,r}))\, C_\theta(\mathbf{u}_t^{(r)}, \mathbf{c}_{t,r}), \qquad r = 1, \ldots, R-1. \tag{16}$$

Here, A_θ applies conditioning-dependent scale and shift after affine-free normalization. C_θ is a conditionally modulated residual SwiGLU module with hidden width $2d$. The recurrent gate is a scalar per example and is broadcast across tokens and channels.

Prediction head. A conditionally modulated normalization layer followed by a linear projection maps the final hidden state back to the diffusion latent space. Only the final recursion pass is decoded and supervised. Hidden states are reset between denoising evaluations; we do not carry hidden states or KV caches across diffusion steps.

C.2 Initialization

All models are trained from scratch. Unless otherwise specified, linear layers and the flattened patch-projection kernel use Xavier-uniform initialization with zero biases. Class embeddings and the learned projections in the time-conditioning network are initialized from $\mathcal{N}(0, 0.02^2)$, while sinusoidal embeddings are fixed.

The final projections of the adaptive-modulation networks, the prediction projection, and the controller output projection are zero-initialized. The recurrent gate is initialized with zero weights and bias -2, corresponding to an initial gate value of $\sigma(-2) \approx 0.119$. This initialization keeps recursive updates small at the beginning of training.

C.3 Image Generation Setup

C.3.1 Controlled DiT Setup with SD-VAE

Latent representation and backbone. For the controlled experiments, we follow the original DiT setting and use a frozen pretrained SD-VAE. Images at 256×256 resolution are encoded to $4 \times 32 \times 32$ latents and scaled by 0.18215. A 2×2 patch size produces 256 tokens. Unless otherwise specified, the backbone uses $L = 3$ unique Transformer layers with hidden dimension $d = 768$. This implementation uses LayerNorm and SwiGLU feed-forward layers, without rotary positional embeddings or query–key normalization.

Diffusion objective. We use a discrete diffusion process with 1,000 noise levels and a linear noise schedule. For clean latent $\mathbf{z}_0$, timestep t, and $\epsilon \sim \mathcal{N}(0, I)$,

$$\mathbf{z}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{z}_0 + \sqrt{1 - \bar{\alpha}_t}\,\epsilon. \tag{17}$$

The model predicts the added noise and is trained with

$$\mathcal{L}_{\mathrm{SD}} = \mathbb{E}\left[\left\|\epsilon_\theta(\mathbf{z}_t, t, y) - \epsilon\right\|_2^2\right]. \tag{18}$$

Only the prediction from the final recursion pass is supervised. We use fixed diffusion variance and do not train a variance-prediction head.

Optimization. We use AdamW with learning rate 10^{-4}, $(\beta_1, \beta_2) = (0.9, 0.999)$, and zero weight decay. The global batch size is 256 and gradients are clipped to norm 1. Training uses bfloat16 mixed precision and an EMA of the model parameters with decay 0.9999.

C.3.2 VA-VAE Setup

Latent representation and backbone. For our primary scaling experiments, we use a frozen pretrained VA-VAE (Yao et al., 2025). It produces $32 \times 16 \times 16$ latent tensors, which are standardized using channel-wise statistics computed from the training set. A 1×1 patch size again produces 256 tokens. Unless otherwise specified, recursive models use hidden dimension $d = 1152$, 16 attention heads, RMSNorm, query–key normalization, rotary positional embeddings, and a SwiGLU hidden width of 3072.

Flow-matching objective. We sample $\epsilon \sim \mathcal{N}(0, I)$ and $t = \sigma(\xi)$ with $\xi \sim \mathcal{N}(0, 1)$. The interpolated latent and target velocity are

$$\mathbf{z}_t = (1 - t)\epsilon + t\mathbf{z}_0, \qquad \mathbf{u} = \mathbf{z}_0 - \epsilon. \tag{19}$$

Given prediction $\hat{\mathbf{u}}_\theta = \mathbf{v}_\theta(\mathbf{z}_t, t, y)$, we optimize

$$\mathcal{L}_{\text{VA}} = \mathbb{E}\left[\text{MSE}(\hat{\mathbf{u}}_\theta, \mathbf{u}) + 1 - \text{Cos}(\hat{\mathbf{u}}_\theta, \mathbf{u})\right]. \tag{20}$$

MSE is averaged over all latent elements. Cosine similarity is computed over channels at each spatial location and then averaged spatially. The two terms receive equal weight, and supervision is applied only after the final recursion pass.

Classifier-free conditioning. Class labels are dropped with probability 0.1. A single dropout decision is sampled for each example and reused across all of its recursion passes. Diffusion-time and recursion-step conditioning are always retained.

Optimization. We use AdamW with learning rate 2×10^{-4}, $(\beta_1, \beta_2) = (0.9, 0.95)$, and zero weight decay, without gradient clipping. Models are trained for 80,000 optimizer updates with global batch size 1,024, corresponding to approximately 64 ImageNet epochs. Recursive runs use eight GPUs with 32 examples per GPU and four gradient accumulation steps. Training uses bfloat16 mixed precision and EMA decay 0.9999.

Evaluation. Unless otherwise specified, ImageNet generation metrics are computed from 50K generated samples. We report FID as the primary metric together with IS, sFID, precision, and recall. For compute-scaling comparisons, we additionally report the number of unique model parameters and measured inference FLOPs.

D Experimental Details

D.1 Analysis of Depth and Diffusion Time

Evaluation setup. We analyze independently trained single-pass ($R = 1$) DiTs with 7, 14, 21, and 28 layers, using EMA checkpoints at 80K training updates. All models have hidden width 1152, 16 attention heads, and a common VA-VAE representation with latent shape $32 \times 16 \times 16$. The training-update count is matched; parameter counts and training FLOPs increase with depth.

We sample 2,048 images uniformly without replacement from the ImageNet validation set. Images are center-cropped to 256×256 and encoded using one sampled VA-VAE posterior latent per image, normalized with the training latent statistics. Each image is paired with one Gaussian noise tensor. The same latents, noise, and class labels are reused across all models, timesteps, and interventions. Evaluation uses FP32 with TF32 enabled and no classifier-free guidance.

Diffusion time follows $t = 1$ for noise and $t = 0$ for data. For each clean latent z_i and noise ϵ_i, we construct $x_{t,i} = (1-t)z_i + t\epsilon_i$ and evaluate at

$$t \in \{0.98, 0.95, 0.90, 0.80, 0.70, 0.60, 0.50, 0.40, 0.30, 0.20, 0.10, 0.02\}.$$

These are fixed diagnostic times; no timestep sampling is used during evaluation. Expressing velocity in this time convention, the prediction error of an L-layer model is

$$\mathcal{E}_L(t) = \frac{1}{ND} \sum_{i=1}^{N} \|\hat{v}_L(x_{t,i}, t, y_i) - (\epsilon_i - z_i)\|_2^2, \tag{21}$$

where $N = 2048$ and $D = 32 \cdot 16 \cdot 16$. The implementation uses the reversed coordinate $s = 1 - t$; reversing both the predicted and target velocities leaves the MSE unchanged. We report the MSE component separately from the cosine term used during training.

Stage sensitivity. For Figure 3b, we partition the 28-layer model into four stages comprising blocks 1–7, 8–14, 15–21, and 22–28. For one stage at a time, we replace its output with

$$\tilde{h}_{\text{out}} = h_{\text{in}} + \alpha(h_{\text{out}} - h_{\text{in}}), \tag{22}$$

then execute the remaining layers and the trained prediction head. The main figure uses $\alpha = 0.8$, corresponding to a 20% attenuation of the aggregate stage update. We also evaluate $\alpha \in \{0.9, 1.1, 1.2\}$ as controls.

For stage g, the plotted quantity is

$$S_g(t) = 100\, \frac{\mathcal{E}_{g,0.8}(t) - \mathcal{E}_{28}(t)}{\mathcal{E}_{28}(t)}. \tag{23}$$

Table 3: **Detailed recursion-module ablations.** We report FID, sFID, Inception Score (IS), Precision, and Recall for different recursion depths under CFG scales 1.0 and 1.5. Lower is better for FID and sFID; higher is better for the remaining metrics. "–" denotes configurations not evaluated.

		CFG = 1.0					CFG = 1.5				
Design	R	**FID** ↓	**sFID** ↓	**IS** ↑	**Pre.** ↑	**Rec.** ↑	**FID** ↓	**sFID** ↓	**IS** ↑	**Pre.** ↑	**Rec.** ↑
A	1	91.12	15.82	14.91	0.293	0.439	61.41	11.32	26.09	0.400	0.447
	2	82.00	12.23	16.30	0.330	0.492	53.92	8.79	28.30	0.445	0.507
	4	59.63	8.47	22.24	0.404	0.584	35.95	8.76	39.59	0.537	0.549
	8	44.46	7.03	30.08	0.484	0.632	22.12	5.47	62.90	0.637	0.548
B	1	–	–	–	–	–	–	–	–	–	–
	2	75.36	10.12	17.87	0.343	0.537	48.03	7.65	31.29	0.464	0.523
	4	52.40	7.80	25.95	0.444	0.620	27.82	5.82	52.94	0.593	0.555
	8	40.69	6.69	33.87	0.502	0.635	19.24	5.25	70.84	0.669	0.669
C	1	–	–	–	–	–	–	–	–	–	–
	2	71.23	9.74	18.90	0.356	0.550	44.27	7.36	34.38	0.482	0.532
	4	50.70	7.93	26.63	0.450	0.625	26.87	5.83	53.22	0.610	0.553
	8	39.68	6.76	34.19	0.509	0.628	18.97	5.54	70.74	0.671	0.539
D	1	–	–	–	–	–	–	–	–	–	–
	2	68.21	9.89	19.90	0.364	0.548	40.85	7.19	37.73	0.502	0.530
	4	50.13	8.02	27.65	0.456	0.618	25.37	5.74	57.82	0.619	0.539
	8	**38.57**	6.94	35.62	0.523	0.631	**17.01**	5.49	**77.87**	**0.687**	0.528

Positive values indicate that attenuation increases error. The heatmap uses a symmetric logarithmic color scale. Sensitivity varies across both stages and diffusion time: for blocks 15–21, the error increase is approximately zero at $t = 0.9$ and reaches 0.618% at $t = 0.1$. The controls also show that sensitivity depends on perturbation strength; attenuating an update does not uniformly increase error.

Marginal depth benefit. For Figure 3c, we compare adjacent independently trained depths using

$$G_L(t) = \mathcal{E}_L(t) - \mathcal{E}_{L+7}(t), \qquad L \in \{7, 14, 21\}. \tag{24}$$

Each comparison adds seven blocks of the same width. Positive values favor the deeper model. The absolute gains peak near $t = 0.1$ for all three comparisons and decrease at $t = 0.02$. Relative reductions, $100G_L(t)/\mathcal{E}_L(t)$, likewise show larger benefits in the middle and late portions of denoising than at $t = 0.9$. These comparisons measure depth benefits across separately trained networks, rather than additional passes through shared weights.

Statistical analysis. We use 2,000 paired bootstrap resamples over images, preserving each image's pairing across models, times, and interventions. The displayed 95% simultaneous confidence intervals use the bootstrap maximum absolute standardized deviation over 48 stage–time comparisons in panel (b) and 36 depth–time comparisons in panel (c), treated as separate families. Hatching in panel (b) marks intervals containing zero; shading in panel (c) denotes the confidence bands.

D.2 Recursion-Aware Design Ablations

We provide the full results for the recursion-aware design ablations from Section 4.3. We compare (A) standard diffusion-time conditioning, (B) 2D time embedding over (t, r), (C) recursion-aware modulation, and (D) the full gated residual design.

Across both guidance settings, explicitly incorporating recursion progress improves recursive scaling. At $R = 8$ and CFG $= 1.5$, FID decreases from 22.12 for (A) to 19.24, 18.97, and 17.01 for (B)–(D), respectively. The improvements are also reflected in IS and precision, while individual metrics such as recall and sFID do not improve monotonically for every design component. Overall, these results support the full recursion-aware formulation used in our main experiments.

D.3 TIME-ADAPTIVE RECURSION

Inference schedule. Time-adaptive recursion allocates greater recursion depth to later denoising evaluations while preserving the total number of recursion passes per sampling trajectory. Let $M \geq 2$ denote the number of model evaluations, ordered from noise to clean, and let $\bar{R}$ be the target average recursion depth, with $M\bar{R}$ integral. We first construct the real-valued target depths

$$\widetilde{R}_j = \frac{\bar{R}}{2} + \bar{R}\frac{j}{M-1}, \qquad j = 0, \ldots, M-1. \tag{25}$$

We initialize $R_j = \lfloor \widetilde{R}_j \rfloor$ and distribute the remaining $M\bar{R} - \sum_j \lfloor \widetilde{R}_j \rfloor$ passes by incrementing the entries with the largest fractional parts, breaking ties in favor of earlier evaluations. The resulting integer depths are nondecreasing and satisfy

$$\sum_{j=0}^{M-1} R_j = M\bar{R}. \tag{26}$$

All depths are at least one for the settings considered. Here, M counts actual model evaluations; an Euler grid with N time points entails $M = N - 1$ evaluations. The sampling grid and numerical solver are otherwise unchanged.

Training depth assignment. We construct a deterministic, piecewise-constant timestep-to-depth mapping $R(t)$ from a fixed reference inference schedule. Let τ_j and R_j^{ref} denote its model-evaluation times and assigned depths, respectively. Whenever $R_{j-1}^{\text{ref}} \neq R_j^{\text{ref}}$, we place a region boundary at

$$b_j = \frac{\tau_{j-1} + \tau_j}{2}. \tag{27}$$

Each region inherits the corresponding reference depth, with the first and last regions extending to the endpoints of the timestep domain. This mapping remains fixed throughout training. When the evaluation grid changes, we reconstruct the inference depth schedule for its actual number of model evaluations; the training mapping remains unchanged.

For each training example, we independently sample t_i from the original logit-normal distribution and then assign $R_i = R(t_i)$. Depth assignment and batch compaction therefore preserve the original joint IID timestep distribution. The average training depth is determined by $\mathbb{E}_t[R(t)]$ and need not equal $\bar{R}$; the exact budget constraint above applies to inference trajectories.

Variable-depth execution. All examples execute the first recursion pass. After pass r, examples with $R_i = r$ produce their final prediction and leave the active batch, while those with $R_i > r$ continue using their updated hidden states. Each example retains the same diffusion timestep and class-dropout mask throughout its recursion passes. Predictions are restored to the original batch order, and the original per-example loss is evaluated once on each example's final prediction and averaged over the original batch. Thus, each example contributes equally to the batch mean regardless of its assigned depth. Gradients propagate through all executed recursion passes without detaching hidden states.

We provide additional results for the time-adaptive recursion study in Section 4.4. We divide the denoising trajectory into four consecutive regions from high to low noise and assign each region a recursion depth (R_1, R_2, R_3, R_4). All schedules use the same total recursion budget, $\sum_i R_i = 24$, allowing us to isolate the effect of where recursive computation is allocated.

The same trend observed under CFG $= 1.5$ persists without additional guidance. Under CFG $= 1.0$, the uniform schedule $(6, 6, 6, 6)$ achieves FID 37.37 and IS 37.16, while shifting computation toward later denoising stages improves these to FID 35.37 and IS 40.00 with $(1, 7, 8, 8)$. Conversely, the early-heavy schedule $(8, 6, 6, 4)$ degrades to FID 40.75 and IS 34.29. These results further support allocating more recursive computation to later, lower-noise stages of the denoising trajectory.

D.4 LANGUAGE DIFFUSION

Datasets. **Text8** (Mahoney, 2011) contains character-level English Wikipedia text. We use the standard 90M/5M/5M-character training/validation/test split, with 27 characters and an additional mask token. Training uses 256-character crops sampled within 512-character chunks.

Table 4: **Time-adaptive recursion under CFG $= 1.0$.** Recursion is allocated across four diffusion regions from high to low noise under the same total recursion budget.

Schedule	FID↓	IS↑	sFID↓	Pre.↑	Rec.↑
$(8, 6, 6, 4)$	40.75	34.29	7.11	0.499	0.631
$(6, 6, 6, 6)$	37.37	37.16	**6.74**	0.521	0.640
$(4, 6, 6, 8)$	36.78	38.18	7.13	0.524	0.641
$(2, 6, 8, 8)$	35.71	39.26	6.96	0.530	**0.647**
$(1, 7, 8, 8)$	**35.37**	**40.00**	6.98	**0.531**	**0.647**

For **LM1B** (Chelba et al., 2013), we follow the MDLM preprocessing convention (Sahoo et al., 2024), using its detokenization procedure and the `bert-base-uncased` tokenizer. Sentences are padded or truncated to 128 positions without sentence packing; padding is excluded from the loss.

Objective and architecture. We adopt the continuous-time masked diffusion objective with SUBS parameterization (Sahoo et al., 2024). We draw stratified timesteps $t \in [10^{-3}, 1]$, mask tokens with probability $0.999t$, and weight masked-token cross-entropy by $1/t$, normalizing over all nonpadding tokens. The backbone contains three Transformer blocks with 12 attention heads and hidden dimension 384 for Text8 or 768 for LM1B. We apply the shared blocks $R \in \{1, 2, 4, 8\}$ times per denoiser evaluation, keeping R fixed across diffusion timesteps. The $R = 1$ model is a plain three-block baseline. Recursive variants use discrete recursion-index conditioning, a conditioned SwiGLU controller, and gated residual hidden-state updates, with full backpropagation through all passes and no step-size scaling. Explicit diffusion-time conditioning is disabled.

Training. All models use global batch size 512 and seed 1. Text8 models train for 100,000 updates, processing 13.1072B character positions. LM1B models train for 177,543 updates, corresponding to three epochs and 2.767B nonpadding tokens. Comparisons therefore match training updates and data exposure within each dataset; computational cost increases with recursion. We use AdamW with learning rate 3×10^{-4}, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, and zero weight decay. The learning rate increases linearly over 2,500 updates and remains constant thereafter. We use dropout 0.1, gradient clipping at norm 1, BF16 mixed precision, and EMA decay 0.9999.

Evaluation. We report negative-ELBO estimates as bits per character (BPC) on Text8 and exponentiated per-token NLL (perplexity) on LM1B. These are likelihood evaluations and do not involve iterative text generation; LM1B test perplexity is distinct from generative perplexity scored by an external language model.

For Text8, raw and EMA train/validation BPC are evaluated every 2,000 updates using two corruption repeats. Training evaluation uses 4,096 fixed windows. Validation and test evaluation each use 19,531 nonoverlapping 256-character windows, covering 4,999,936 characters and omitting the final 64 characters of each split. We select the EMA checkpoint with the lowest validation BPC and evaluate test BPC using eight corruption repeats. All selected checkpoints are at update 100,000.

For LM1B, we reserve the first 4,096 official test examples for monitoring and use the remaining 302,592 examples for final reporting. Raw-weight monitoring is performed every 5,000 updates with two corruption repeats. Final raw and EMA checkpoints at update 177,543 are evaluated with four corruption repeats using seeds 2026–2029. All experiments use one training seed; corruption repeats measure evaluation noise rather than training-seed variability.

Further results. Table 5 and Table 6 report additional metrics for fixed recursion. Increasing recursion improves held-out likelihood while largely preserving parameter count for $R \geq 2$. The increase from $R = 1$ primarily reflects the additional recurrent controller.

D.5 Molecular Diffusion

Experimental setup. We evaluate unconditional molecular generation on MOSES using the DiGress graph transformer with DAE atom encoding and HDDM transitions. The representation contains 12 atom types, four atom-group states, and one mask state; bonds use uniform diffusion. The

Table 5: Text8 results at 100,000 updates. BPC is ELBO-based; lower is better. Train BPC is evaluated on a fixed training subset.

R	Params.	Raw BPC		EMA BPC		
		Train	Valid	Train	Valid	Test
1	6.38M	1.9128	1.8866	1.9023	1.8776	1.9232
2	7.46M	1.7754	1.7436	1.7646	1.7342	1.7847
4	7.46M	1.6821	1.6502	1.6679	1.6368	1.6911
8	7.47M	1.6380	1.6054	1.6236	1.5908	**1.6478**

Table 6: LM1B results at 177,543 updates. NLL and perplexity are ELBO-based estimates on the final reporting split; lower is better.

R	Params.	Raw		EMA	
		NLL	PPL	NLL	PPL
1	70.19M	3.8383	46.45	3.7715	43.44
2	74.13M	3.7153	41.07	3.6425	38.19
4	74.13M	3.6487	38.43	3.5712	35.56
8	74.13M	3.6225	37.43	3.5426	**34.56**

recursive backbone contains three shared transformer layers, with hidden dimensions of 256, 64, and 128 for atom, bond, and global features, respectively, and eight attention heads. Each denoiser call applies these layers R times, including the initial pass. Subsequent passes use gated residual updates conditioned on diffusion time and recursion index.

Training. We train separate models with fixed recursion depths $R \in \{1, 2, 4, 8\}$ for 50 epochs on MOSES dataset (Polykovskiy et al., 2020). All runs use AdamW with learning rate 2×10^{-4}, 10,000 warmup updates followed by cosine decay, weight decay 10^{-12}, and gradient clipping at 2. Training uses six RTX A5000 GPUs, BF16 precision, and an effective global batch size of 1,536. We retain the original molecular denoising loss and use the final-epoch EMA weights with decay 0.999.

Sampling and metrics. We evaluate $T \in \{25, 50, 100, 200, 500\}$ sampling steps, generating 10K molecules per setting. We test both on original posterior sampling and PNJ samples following the implementation in Jung et al. (2026). Temperature and top-p are both 1. Reference-based metrics use the implementation's MOSES validation reference set. We report validity, Fréchet ChemNet Distance (FCD; Preuer et al. 2018), nearest-neighbor similarity (SNN), scaffold similarity, molecular quality, mean QED (Bickerton et al., 2012), and mean synthetic accessibility (SA; Ertl and Schuffenhauer 2009). Following Lee et al. (2025), we define molecular quality as the number of unique valid molecules satisfying QED ≥ 0.6 and SA ≤ 4, divided by all generated samples. Lower FCD and SA are better; higher values are preferred for the remaining metrics.

Fixed-recursion results. Table 7 reports the complete results using the PNJ sampler. At every sampling budget, increasing R improves validity, FCD, SNN, molecular quality, QED, and SA. Scaffold similarity instead peaks at $R = 4$ for all five budgets. We observe more sampling steps do not improve every metric: for $R = 8$, FCD is lowest at $T = 100$ (1.20), whereas validity and molecular quality continue to improve through $T = 500$.

Adaptive recursion. We also train models with average recursion budgets $R \in \{2, 4, 8\}$, allocating more passes toward the clean end of diffusion. During training, each molecule receives an independent timestep $t \sim \mathcal{U}[10^{-4}, 1)$, where $t = 1$ denotes noise, and uses

$$R(t) = \text{round}\left(\frac{3R}{2} - Rt\right).$$

The resulting depth ranges are 1–3, 2–6, and 4–12. At inference, we discretize a linear ramp from $R/2$ to $3R/2$ over the T denoiser calls using largest-remainder rounding, so that the total number of passes is exactly RT. Thus, fixed and adaptive recursion use the same number of transformer applications at each matched (R, T) setting.

Table 7: **Full molecular generation results on MOSES.** We compare fixed and time-adaptive recursion across sampling depths T. For adaptive recursion, R denotes the average recursion depth under the linear schedule from $R/2$ to $3R/2$. Bold indicates the best result within each sampling budget.

T	Type	R	Validity ↑	FCD ↓	SNN ↑	Scaffold ↑	Quality ↑	QED ↑	SA ↓
25	Fixed	1	40.9	15.82	0.311	0.018	12.9	0.729	4.35
	Fixed	2	55.6	7.79	0.378	0.118	30.4	0.755	3.65
	Fixed	4	77.2	2.96	0.434	**0.183**	57.8	0.777	3.07
	Fixed	8	82.8	1.64	0.456	0.161	68.6	0.789	2.88
	Adaptive	2	60.7	6.60	0.387	0.110	35.3	0.765	3.58
	Adaptive	4	83.4	2.13	0.449	0.165	67.0	0.786	2.95
	Adaptive	8	**85.9**	**1.58**	**0.463**	0.164	**73.5**	**0.790**	**2.80**
50	Fixed	1	47.4	16.29	0.311	0.023	14.4	0.733	4.36
	Fixed	2	62.9	6.91	0.390	0.076	35.7	0.763	3.58
	Fixed	4	85.5	2.15	0.454	**0.185**	67.5	0.784	2.93
	Fixed	8	89.0	1.31	0.473	0.157	77.3	0.791	2.76
	Adaptive	2	68.6	5.61	0.402	0.136	42.8	0.777	3.48
	Adaptive	4	89.3	1.59	0.465	0.174	74.5	0.786	2.83
	Adaptive	8	**91.4**	**1.20**	**0.481**	0.172	**81.5**	**0.796**	**2.69**
100	Fixed	1	51.5	16.45	0.309	0.020	15.1	0.734	4.39
	Fixed	2	67.3	5.90	0.405	0.112	41.3	0.772	3.45
	Fixed	4	89.8	1.79	0.470	0.165	74.0	0.787	2.83
	Fixed	8	92.0	1.20	0.489	0.134	81.3	0.795	2.68
	Adaptive	2	72.9	4.73	0.418	0.131	47.4	0.781	3.36
	Adaptive	4	92.9	1.25	0.482	0.140	80.1	0.792	2.73
	Adaptive	8	**94.1**	**1.12**	**0.494**	**0.180**	**85.9**	**0.800**	**2.62**
200	Fixed	1	52.4	16.95	0.310	0.013	14.7	0.735	4.44
	Fixed	2	70.2	5.55	0.416	0.151	44.2	0.772	3.39
	Fixed	4	92.5	1.66	0.481	**0.201**	77.9	0.789	2.75
	Fixed	8	93.8	1.40	0.498	0.124	83.3	0.792	2.64
	Adaptive	2	76.3	4.14	0.433	0.158	52.6	0.786	3.25
	Adaptive	4	94.8	1.29	0.495	0.155	83.7	0.795	2.66
	Adaptive	8	**96.1**	**1.22**	**0.504**	0.144	**88.4**	**0.799**	**2.57**
500	Fixed	1	56.0	16.61	0.311	0.009	16.5	0.739	4.43
	Fixed	2	74.6	5.00	0.430	0.115	49.0	0.778	3.29
	Fixed	4	94.2	1.70	0.494	**0.154**	80.7	0.791	2.69
	Fixed	8	95.4	1.67	0.509	0.125	85.6	0.794	2.59
	Adaptive	2	80.0	3.77	0.448	0.116	57.8	0.792	3.14
	Adaptive	4	96.9	**1.46**	0.508	0.144	87.3	0.796	2.59
	Adaptive	8	**97.3**	1.47	**0.516**	0.134	**90.2**	**0.802**	**2.53**

Adaptive recursion improves validity, FCD, SNN, and molecular quality at all 15 matched settings. At $R = 8$ and $T = 100$, FCD decreases from 1.20 to 1.12; at $T = 500$, molecular quality increases from 85.61% to 90.20%. Training takes approximately 5–12% longer. These single-seed comparisons also differ in rank-wise random-number streams, epoch-tail padding, and recursion-embedding capacity, so they do not isolate the schedule alone.

Sampler choice. We compare the original posterior sampler and PNJ at $T = 500$, using the same final EMA checkpoints and 10,000 samples per setting. Figure 7 shows that increasing fixed recursion depth improves validity, FCD, and molecular quality under both samplers. The recursion trend therefore holds for both sampling procedures tested.

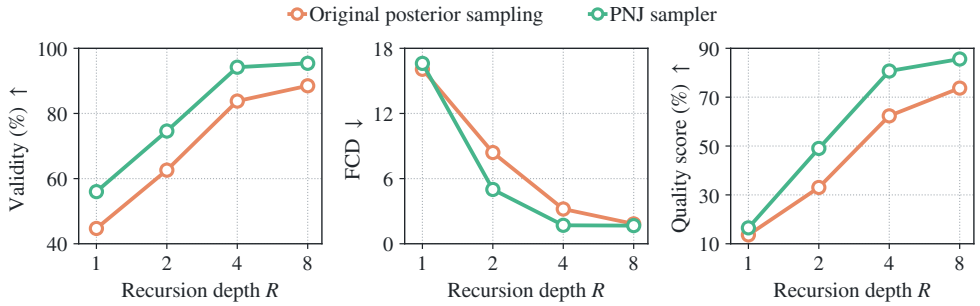


Figure 7: **Recursion benefits both samplers.** Original posterior sampling and PNJ at $T = 500$, using the same checkpoints across $R \in \{1, 2, 4, 8\}$. Increasing recursion improves **(a)** validity, **(b)** FCD, and **(c)** molecular quality under both samplers. Each setting uses 10,000 generated molecules.