Generative Recursive Reasoning

Junyeob Baek[1†*] **Mingyu Jo**[1†*] **Minsu Kim**[1,2]

Mengye Ren[3] **Yoshua Bengio**[2,4] **Sungjin Ahn**[1,3†]

[1]KAIST [2]Mila – Québec AI Institute
[3]New York University [4]Université de Montréal

Abstract

How should future neural reasoning systems implement extended computation? Recursive Reasoning Models (RRMs) offer a promising alternative to autoregressive sequence extension by performing iterative latent-state refinement with shared transition functions. Yet existing RRMs are largely deterministic, following a single latent trajectory and converging to a single prediction. We introduce *Generative Recursive reAsoning Models (GRAM)*, a framework that turns recursive latent reasoning into probabilistic multi-trajectory computation. GRAM models reasoning as a stochastic latent trajectory, enabling multiple hypotheses, alternative solution strategies, and inference-time scaling through both recursive depth and parallel trajectory sampling. This yields a latent-variable generative model supporting conditional reasoning via $p_\theta(y \mid x)$ and, with fixed or absent inputs, unconditional generation via $p_\theta(x)$. Trained with amortized variational inference, GRAM improves over deterministic recurrent and recursive baselines on structured reasoning and multi-solution constraint satisfaction tasks, while demonstrating an unconditional generation capability. https://ahn-ml.github.io/gram-website

1 Introduction

A central question for future neural reasoning systems is how extended computation should be implemented. Large autoregressive models typically scale reasoning by extending a sequence-generation process, whether intermediate computation is expressed explicitly as chain-of-thought tokens or implicitly in hidden or latent representations [1–6]. A complementary direction is explored by Recursive Reasoning Models (RRMs), which use repeated computation to refine a persistent latent state rather than to append new elements to an output or reasoning sequence [7–9]. This approach is appealing because it decouples reasoning depth from both parameter scale and output length: a compact model can perform many steps of internal computation by repeatedly applying shared transition functions over time.

Recent recursive reasoning models such as HRM [8] and TRM [9] provide early evidence for the potential of this approach in structured reasoning. Rather than producing a solution in a single feedforward pass, they perform extended computation through iterative latent-state refinement, deep supervision across refinement steps, and reasoning-oriented recurrent designs such as hierarchical latent dynamics. These features make them well suited to problems requiring constraint propagation, state tracking, iterative correction, and multi-step inference. More broadly, they build on a principle

*Equal contribution
†Correspondence to: Junyeob Baek (wnsdlqjtm@kaist.ac.kr), Mingyu Jo (mingyu.jo@kaist.ac.kr), Sungjin Ahn (sungjin.ahn@kaist.ac.kr)

Preprint.

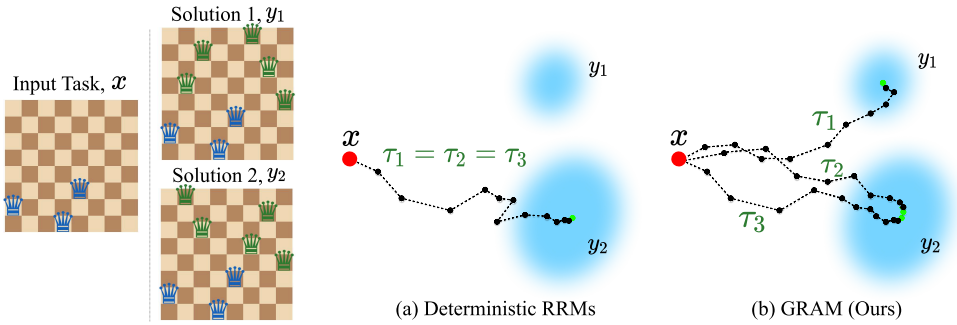


Figure 1: Comparison of Latent Reasoning Trajectories. Left: N-Queens Example with two valid solutions. Right: Given three independent runs for latent reasoning (τ_1, τ_2, τ_3): (a) Prior RRMs (e.g. HRM, TRM) are deterministic—all runs collapse to an identical trajectory, converging to a single solution and failing to explore alternatives, while (b) GRAM explores diverse trajectories, producing diverse trajectories that reach multiple valid solutions y_1 and y_2, while naturally enabling parallel inference-time scaling.

also explored in recurrent Transformer architectures such as Universal Transformers [10] and Looped Transformers [7]: shared Transformer blocks can be repeatedly applied to increase computational depth without increasing parameter count. Together, these models suggest that reasoning capability can emerge not only from scaling model size or generating longer traces, but also from the organization of computation itself.

While recurrent latent-state refinement provides an appealing mechanism for efficiently increasing reasoning depth, depth alone is not sufficient for many reasoning problems. A capable reasoning system should also be able to maintain uncertainty, consider alternative hypotheses, and explore multiple possible solution strategies [11, 12]. This is especially important in settings where ambiguity or multiple valid solutions are intrinsic, and more generally in problems where a single refinement path may become trapped in a suboptimal reasoning trajectory. In this sense, future RRMs should be not only deep, in the sense of repeated refinement, but also wide, in the sense of maintaining and exploring multiple latent trajectories in parallel.

Existing RRMs [7–10], however, remain fundamentally deterministic: given the same input and initialization, they follow a single latent trajectory and converge to a single prediction. This deterministic recursion collapses the space of plausible reasoning paths into a single attractor, leaving probabilistic multi-hypothesis latent reasoning largely unexplored within the RRM paradigm. This motivates the central question of our work: *can recursive latent computation support probabilistic, generative, multi-hypothesis reasoning while preserving the efficiency of compact recurrent models?*

In this paper, we propose Generative Recursive reAsoning Models (GRAM), a framework that turns recursive latent reasoning into probabilistic multi-trajectory computation. GRAM treats the reasoning process itself as a stochastic latent trajectory: at each recursion step, the model samples a transition conditioned on the input and the current reasoning state, rather than deterministically updating to a single next state. Repeating this process defines a distribution over possible reasoning trajectories, allowing the model to maintain multiple hypotheses, explore alternative solution strategies, and scale inference not only by increasing recursive depth but also by sampling trajectories in parallel. From a probabilistic perspective, GRAM is a latent-variable generative model: it models $p_\theta(y \mid x)$ by marginalizing over latent reasoning trajectories, while the same recursive process can also define an unconditional generative model $p_\theta(x)$ when the input is fixed or absent.

We evaluate GRAM on controlled reasoning and generation tasks that serve as probes of the architectural properties targeted by our formulation: recursive refinement, stochastic exploration, multi-solution coverage, and inference-time scaling. Given this goal, our experiments focus on comparisons with the most relevant deterministic recurrent and recursive latent reasoning baselines, including Looped Transformers, HRM, and TRM, rather than frontier-scale general-purpose LLMs whose training data, inference budgets, and external scaffolding are not directly comparable. Sudoku-Extreme [8] and ARC-AGI [13, 14] test structured reasoning under hard constraints and abstract transformations; N-Queens and Graph Coloring evaluate multi-solution recovery; and binarized MNIST [15] probes the unconditional generative interpretation.

Our main contribution is to establish probabilistic multi-trajectory recursion as a design principle for future recurrent and recursive reasoning architectures. Concretely, we make three contributions.

First, we formulate recursive reasoning as a latent-variable generative process, where solutions are obtained by marginalizing over stochastic reasoning trajectories. Second, we introduce width-based inference-time scaling, enabling inference to scale not only with recursive depth but also with the number of sampled latent trajectories. Third, we provide empirical evidence that this formulation yields the intended architectural advantages over deterministic recurrent and recursive baselines, improving structured reasoning, multi-solution constraint satisfaction, and unconditional generation.

2 Generative Recursive Reasoning Models

In this section, we introduce Generative Recursive reAsoning Models (GRAM), an instantiation of probabilistic recursive reasoning. We describe the architecture in Section 2.1 and the training procedure in Section 2.2, with an architecture schematic shown in Figure 2.

2.1 Architecture

Overview. GRAM models the conditional distribution $p_\theta(y \mid x)$ by marginalizing over stochastic latent reasoning trajectories. Given an input x, GRAM first computes an embedding

$$e_x = f_{\text{enc}}(x; \theta), \tag{1}$$

which is reused throughout the entire recursive computation. Starting from a fixed initial latent state z_0, the model evolves the latent state through learned stochastic transitions. The recursive computation is organized into two nested levels: inner and outer loops.

At the inner level, a *latent transition* samples a new latent state conditioned on the previous latent state and the input embedding,

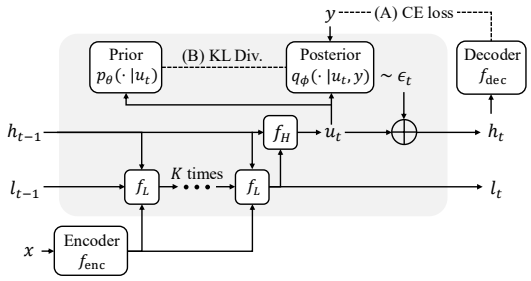


Figure 2: GRAM Architecture. A single stochastic latent transition in the hierarchical instantiation $z = (h, l)$. After K low-level refinements via f_{L}, the high-level update f_{H} produces a deterministic proposal u_t, to which stochastic guidance ϵ_t is added: $h_t = u_t + \epsilon_t$.

$$z_t \sim p_\theta(z_t \mid z_{t-1}, e_x), \qquad t = 1, \ldots, T. \tag{2}$$

At the end of the T transitions, the decoder produces a prediction, $\hat{y} = \arg\max f_{\text{dec}}(z_T; \theta)$. We refer to the sequence of T transitions from the initial state z_0 to the final state z_T as a *supervision step*. A supervision step is the unit at which the decoder is invoked, and the training objective is applied, with gradients computed as described in Section 2.2.

At the outer level, N_{sup} supervision steps are applied recursively, with the final state of one supervision step serving as the initial state of the next, thereby forming the full recursive computation:

$$z_0^{(1)} \xrightarrow{T \text{ transitions}} z_T^{(1)} = z_0^{(2)} \xrightarrow{T \text{ transitions}} \cdots \xrightarrow{T \text{ transitions}} z_T^{(N_{\text{sup}})}, \tag{3}$$

where $z_t^{(n)}$ denotes the latent state at the t-th transition of the n-th supervision step, $z_0^{(1)}$ is the fixed initial state, and the terminal state of one supervision step serves as the initial state of the next ($z_0^{(n+1)} := z_T^{(n)}$). This abstract formulation can be instantiated with various recurrent Transformer backbones, including flat designs such as Universal Transformers and Looped Transformers [10, 7], as well as hierarchical designs such as HRM and TRM [8, 9].

Stochastic Latent Transitions. Unlike prior recursive reasoning models (RRMs) that update the latent state deterministically and follow a single fixed trajectory [8, 9], GRAM defines $p_\theta(z_t \mid z_{t-1}, e_x)$ as a stochastic transition, so that repeated computation induces a distribution over latent reasoning trajectories. Concretely, GRAM realizes this transition as a learned stochastic residual perturbation around a deterministic update: at each transition, the model first computes a deterministic update u_t from z_{t-1} and e_x, then samples a conditional perturbation from a state-dependent Gaussian, and adds it to u_t:

$$\epsilon_t \sim p_\theta(\epsilon_t \mid u_t) := \mathcal{N}\big(\mu_\theta(u_t),\, \sigma_\theta^2(u_t) I\big)\,, \tag{4}$$

$$z_t = u_t + \epsilon_t. \tag{5}$$

We refer to ϵ_t as the **learnable stochastic guidance**. The mean $\mu_\theta(u_t)$ encodes a state-dependent direction in which the trajectory is steered, while the variance $\sigma^2_\theta(u_t)$ controls the amount of exploration. This design allows GRAM to capture uncertainty, prevent convergence to local minima, and support robust exploration of the solution space without discarding the deterministic refinement performed by u_t.

Hierarchical Instantiation. We instantiate the latent state with two interacting components, $z = (h, l)$. The high-level component h is updated once per latent transition and carries abstract reasoning state, while the low-level component l is updated K times within a single transition and carries fine-grained intermediate computation. This decomposition separates the two roles across time scales, with h accumulating slowly across transitions and l refined rapidly within each one.

With this hierarchical multi-scale structure, a single transition $z_{t-1} \to z_t$ is computed as follows. The low-level component is first refined for K updates, with the high-level component held fixed:

$$l_{t,k} = f_{\mathrm{L}}(h_{t-1},\, l_{t,k-1},\, e_x;\, \theta), \qquad k = 1, \ldots, K, \tag{6}$$

where $l_{t,0} := l_{t-1}$ and we write $l_t := l_{t,K}$ for the refined low-level component. The high-level component is then updated as a stochastic transition conditioned on the refined l_t,

$$u_t = f_{\mathrm{H}}(h_{t-1},\, l_t;\, \theta), \tag{7}$$

$$\epsilon_t \sim p_\theta(\epsilon_t \mid u_t) := \mathcal{N}\big(\mu_\theta(u_t),\, \sigma^2_\theta(u_t)\, I\big), \tag{8}$$

$$h_t = u_t + \epsilon_t, \tag{9}$$

and we set $z_t = (h_t, l_t)$. Note that stochasticity is introduced only at the high level: the low-level refinement is fully deterministic, while the stochastic guidance signal ϵ_t acts on the slower, more abstract component of the latent state, where it can steer the overall reasoning trajectory across transitions[3]. Under this instantiation, the decoder reads only the high-level component, i.e., $f_{\mathrm{dec}}(z_T) = f_{\mathrm{dec}}(h_T)$. Additional architectural details are provided in Appendix B.1.

Modeling Unconditional Distribution. While the description so far focuses on the conditional setting $p_\theta(y \mid x)$, the same recursive process can also be defined as an unconditional generative model $p_\theta(x)$ when the input is replaced with an empty conditioning embedding. We use this formulation for generation tasks in Section 4.3.

2.2 Training

GRAM is trained to model the conditional distribution $p_\theta(y \mid x)$, where each training example consists of an input x and its corresponding target y. As a probabilistic model, GRAM adopts a latent-variable formulation and is optimized by maximizing an evidence lower bound (ELBO) with respect to the generative parameters θ and variational parameters ϕ.

Latent Variable Modeling. We model GRAM as a latent-variable probabilistic model p_θ, where the full latent trajectory $\tau = (z_0 \to \cdots \to z_{T_{\mathrm{Total}}})$ consists of a sequence of latent variables, with $T_{\mathrm{Total}} = T \times N_{\mathrm{sup}}$. The conditional likelihood is defined as

$$p_\theta(y \mid x) = \int p_\theta(y \mid \tau, x)\, p_\theta(\tau \mid x)\, d\tau, \tag{10}$$

where x denotes the input problem and y denotes the corresponding ground-truth output.

Direct maximum likelihood estimation of $\log p_\theta(y \mid x)$ is intractable due to the marginalization over latent trajectories. We therefore introduce a variational posterior $q_\phi(\tau \mid x, y)$ and optimize the evidence lower bound (ELBO), jointly training θ and ϕ via variational inference:

$$\log p_\theta(y \mid x) \geq \mathbb{E}_{q_\phi(\tau \mid x, y)}[\log p_\theta(y \mid \tau, x)] - \mathrm{KL}(q_\phi(\tau \mid x, y) \,\|\, p_\theta(\tau \mid x))\,. \tag{11}$$

During training, latent trajectories are sampled from the variational posterior $q_\phi(\cdot \mid x, y)$, which has access to both the input problem x and the target output y. At inference time, where y is unavailable, trajectories are instead generated from the learned prior $p_\theta(\cdot \mid x)$.

[3]We also tried injecting noise into the low-level state, but found that it did not improve performance.

Both the prior and the posterior are modeled as conditional Markov processes over latent states:

$$p_\theta(\tau \mid x) = p(z_0) \prod_{t=1}^{T_{\text{Total}}} p_\theta(z_t \mid z_{t-1}, x), \qquad q_\phi(\tau \mid x, y) = p(z_0) \prod_{t=1}^{T_{\text{Total}}} q_\phi(z_t \mid z_{t-1}, x, y). \quad (12)$$

Here, z_0 is a fixed initial state shared by the prior and posterior. Both transitions are implemented by adding reparameterized Gaussian noise ϵ_t after a deterministic update u_t; the posterior uses the same transition module as the prior, but samples from a target-conditioned noise distribution $q_\phi(\epsilon_t \mid u_t, y)$, whereas the prior uses $p_\theta(\epsilon_t \mid u_t)$.

Since the two processes share the same Markov structure and all stochasticity is introduced through $\epsilon_{1:T_{\text{Total}}}$, their trajectory distributions can be equivalently represented in noise space. Moreover, since GRAM decodes the output only from the terminal latent state, the likelihood term satisfies $p_\theta(y \mid \tau, x) = p_\theta(y \mid z_{T_{\text{Total}}}, x)$. Therefore, the full trajectory-level ELBO can be written as

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_\phi}\big[\log p_\theta(y \mid z_{T_{\text{Total}}}, x)\big] - \sum_{t=1}^{T_{\text{Total}}} \mathbb{E}_{q_\phi(\epsilon_{<t} \mid x, y)}\Big[\mathrm{KL}(q_\phi(\epsilon_t \mid u_t, y) \,\|\, p_\theta(\epsilon_t \mid u_t))\Big]. \quad (13)$$

Here, $u_t = f_{\text{H}}(h_{t-1}, l_t)$ denotes the deterministic high-level update before noise injection, as defined in Equation (9). Since u_t depends on h_{t-1}, which is determined by the previously sampled noise variables $\epsilon_{<t} := (\epsilon_1, \ldots, \epsilon_{t-1})$, the expectation averages over these ancestral samples.

Practical Implementation. In practice, following previous recursive reasoning models [8, 9], we train GRAM with deep supervision over N_{sup} consecutive supervision steps, each consisting of T recursive latent transitions. This provides dense learning signals along the full latent trajectory, rather than supervising only the final state after $T_{\text{Total}} = T \times N_{\text{sup}}$ transitions. The terminal state of each step is reused as the initial state of the next step.

Following standard practice for recurrent models with long computation chains, we apply truncated gradient propagation [16, 17], as used in recent recursive reasoning models [8, 9, 18]. In our implementation, gradients are propagated only through the final transition of each supervision step, $z_{T-1}^{(n)} \to z_T^{(n)}$. This gives the following surrogate objective for each supervision step:

$$\mathcal{L}_{\text{GRAM}}^{(n)}(x, y; \theta, \phi) = \mathbb{E}_{q_\phi}\big[\log p_\theta(y \mid z_T^{(n)}, x)\big] - \mathrm{KL}\big(q_\phi(\epsilon_T^{(n)} \mid u_T^{(n)}, y) \,\|\, p_\theta(\epsilon_T^{(n)} \mid u_T^{(n)})\big), \quad (14)$$

where $z_T^{(n)}$ is the terminal state of the current supervision step n, and gradients are stopped through preceding states. Thus, $\mathcal{L}_{\text{GRAM}}$ should be viewed as a truncated surrogate objective rather than the exact ELBO; it introduces a biased but memory-efficient approximation to the full ELBO. Further analysis of this approximation is provided in Appendix A.3, and detailed training hyperparameters are listed in Appendix B.2.

2.3 Inference-Time Scaling

GRAM supports two complementary axes of inference-time scaling: *depth*, by varying the number of recursive transitions, and *width*, by sampling multiple latent reasoning trajectories in parallel. For depth, we follow prior recursive reasoning models [8, 9] in adopting adaptive computation time (ACT) [8–10], which allows each trajectory to terminate at a learned halting depth (details in Appendix A.1). For width — the focus of this section — we draw $\{\tau^{(i)}\}_{i=1}^{N} \sim p_\theta(\tau \mid x)$ from the learned prior and decode each terminal state into a candidate output $\hat{y}^{(i)} = f_{\text{dec}}(z_T^{(i)})$, exploring multiple stochastic reasoning paths simultaneously rather than extending a single trajectory.

To select among candidates, we use either majority voting or best-of-N with a Latent Process Reward Model (LPRM). The LPRM is a value head $v_\psi(z_t)$ trained to predict the final quality of a trajectory from its latent state, using a regression target $r \in [0, 1]$ given by the final prediction accuracy. At inference time, majority voting selects the most frequent prediction, whereas LPRM-guided selection chooses the candidate with the highest predicted terminal value. Details of LPRM training are provided in Appendix A.2. Overall, this procedure improves robustness and solution quality through parallel exploration, without increasing the sequential recursion length.

3 Related Work

Latent Reasoning. Latent reasoning aims to reduce the inefficiency and verbosity of explicit Chain-of-Thought (CoT) by shifting part or all of the reasoning process into latent or continuous

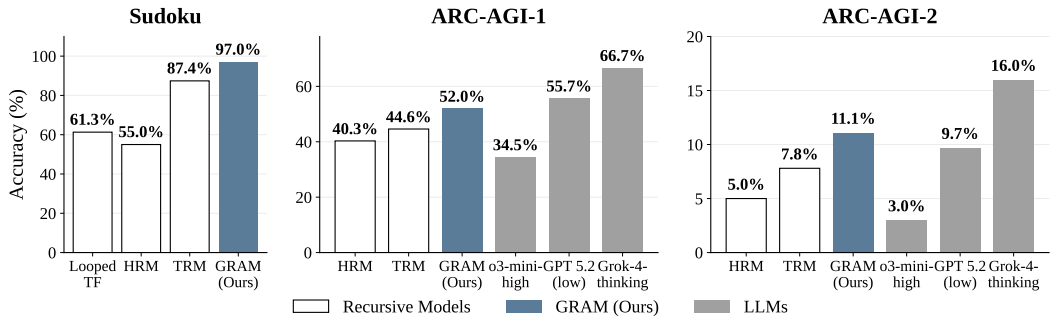


Figure 3: Performance on puzzle benchmarks. On both Sudoku-Extreme and ARC-AGI, GRAM consistently outperforms all deterministic recursive baselines (Looped TF, HRM, TRM), demonstrating that stochastic latent transitions yield substantial gains within the recursive-reasoning paradigm. Looped TF results on ARC-AGI are omitted due to prohibitive training cost (see Section C.1.1) Note that large reasoning model scores are included only as external reference points for benchmark difficulty.

representations [1–6]. By avoiding token-by-token generation of intermediate steps, such representations can make reasoning traces more compact and reduce generation overhead. Existing approaches instantiate this idea through hidden states, latent or soft tokens, continuous thoughts, internal reasoning traces, and recursive state updates for scaling test-time computation [4, 7, 19–23, 18, 24–26]. However, many remain organized around autoregressive sequence generation, where additional computation is tied to generating more tokens, latent positions, or sequential reasoning states.

Recursive Architectures. Recursive architectures perform iterative state updates and have evolved from RNNs to weight-sharing Transformers with adaptive computation [7, 10, 27–32, 25]. Recent recursive reasoning models show that increasing inference-time depth can outperform larger static models [8, 9, 18, 24]. GRAM builds on this line but formulates recurrence as a probabilistic process: instead of following a single deterministic refinement path, it maintains stochastic latent trajectories, enabling multi-path exploration and generative sampling.

Probabilistic Latent State-Space Models. Probabilistic recurrent models use stochastic latent transitions to capture uncertainty and multimodal dynamics, often trained with variational inference [33–38]. They have been widely used in sequential generative modeling, video prediction, and model-based reinforcement learning. GRAM shares this latent state-space view but reinterprets stochastic dynamics as computation rather than temporal observation modeling: latent transitions define possible reasoning trajectories, supporting multi-hypothesis exploration and both conditional $p_\theta(y \mid x)$ and unconditional $p_\theta(x)$ generation.

4 Experiments

GRAM is designed as an architecture for probabilistic recursive reasoning, rather than as a general-purpose large language reasoning model whose training data, inference budgets, prompting strategies, tool use, and external scaffolding are not directly comparable. Following prior work on recurrent and recursive reasoning models [8, 9], we therefore evaluate GRAM on standard structured reasoning tasks that probe the computational properties targeted by our formulation: iterative latent refinement, stochastic trajectory exploration, multi-solution coverage, and inference-time scaling.

In the following, we first evaluate structured reasoning performance on Sudoku-Extreme and ARC-AGI (Section 4.1). We then assess multi-solution behavior on N-Queens and Graph Coloring (Section 4.2). Next, we examine the unconditional generative interpretation of GRAM on binarized MNIST (Section 4.3). Finally, we perform ablation studies to evaluate the impact of key design choices (Section 4.4).

4.1 Challenging Puzzle Tasks

Setup. We evaluate on Sudoku-Extreme [8], which contains 9×9 puzzles with minimal clues requiring extensive constraint propagation, and ARC-AGI Challenge [13, 14], which tests abstract visual reasoning through few-shot pattern recognition. We compare against direct prediction (Trans-

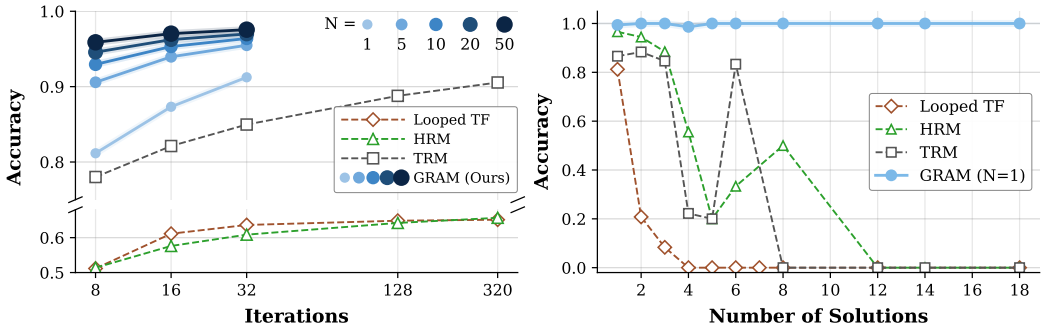


Figure 4: (Left) Inference-time scaling on Sudoku-Extreme. While both TRM and GRAM benefit from longer recursion (x-axis), GRAM additionally scales with parallel sampling (N = number of samples); each iteration corresponds to a supervision step, while meaning $K\times$ more flat iterations in Looped TF. **(Right) Accuracy across number of solutions in N-Queens** (8×8)**.** Conventional deterministic recursive models suffer a sharp performance drop as the number of possible solutions increases, whereas GRAM maintains consistent performance.

former [39]), a flat recursive baselines (Looped TF [7], HRM [8], TRM [9]). Reported large reasoning model results [40] are included as external reference points for benchmark difficulty, rather than as controlled baselines, since their training and inference settings are not directly comparable to task-specific recursive models. For the scaling analysis, all baselines (Looped TF, HRM, TRM) are reproduced under identical settings following Yang et al. [7] and Jolicoeur-Martineau [9].

Stochastic Guidance Improves Reasoning. Figure 3 and Table 8 summarize our main results. GRAM consistently outperforms prior recursive models across all benchmarks. We attribute this improvement to the fundamental difference in how reasoning trajectories are utilized. While Looped TF, HRM, and TRM are restricted to learning from a single deterministic path, GRAM leverages stochastic transitions to explore diverse reasoning trajectories. By training on this richer distribution of solution paths, GRAM acquires more robust reasoning capabilities, allowing it to navigate complex problem spaces more effectively than models constrained to a single sequential refinement process. Detailed experiment results, including more state-of-art methods, are provided in Appendix D.1.

Parallel Sampling Provides a New Test-time Scaling Axis. Figure 4 (left) shows that increasing the number of parallel samples consistently improves performance across all iteration counts. Notably, GRAM with $N = 20$ samples at 16 iterations outperforms all deterministic baselines at 320 iterations, including TRM (97.0% vs 90.5%), despite comparable computational budget. While deterministic recursive models scale only through sequential refinement, GRAM leverages stochastic transitions to explore multiple reasoning paths in parallel. To select the best trajectory, we employ a Latent Process Reward Model (LPRM) that predicts output correctness (Section 2.3). This parallel scaling bypasses the latency bottlenecks of depth-based scaling while achieving superior performance. Additional analysis on the ARC-AGI Challenge is provided in Appendix D.2.

4.2 Multi-solution Puzzle Tasks

Setup. To evaluate whether GRAM can capture diverse solutions, we test on N-Queens (8×8, 10×10) and Graph Coloring (8-vertex, 10-vertex) tasks, where multiple valid solutions exist for each input. We compare against direct prediction (Transformer [39]), recursive models (Looped TF [7], HRM [8], TRM [9]), and generative models (Autoregressive Transformer (AR), MDLM [41]). For N-Queens, we report accuracy (whether the output satisfies all constraints) and coverage (found / total valid solutions, with 20 samples). For Graph Coloring, we report conflict edges (number of constraint violations; lower is better) instead of accuracy. Detailed configurations are provided in Appendix C.2.

Deterministic Recursion Fails on Multi-Solution Tasks. Table 1 reveals that deterministic recursive models structurally cannot capture multiple solutions, with coverage at most 36.1% across all tasks. Figure 4 (right) further illustrates this limitation: as the number of valid solutions increases, all three deterministic recursive baselines exhibit sharp accuracy degradation, whereas GRAM maintains consistent performance regardless of solution count. This confirms that deterministic latent updates

Table 1: Evaluation on N-Queens and Graph Coloring benchmarks. Rec. and Gen. indicate whether the model uses recursive computation and generative sampling, respectively. Values are mean ± standard deviation over runs. Accuracy: single-sample (%). Conflict: constraint-violating edges (↓). Coverage: unique valid solutions discovered with 20 samples (%).

				N-Queens				Graph Coloring			
				8×8		10×10		8-vertex		10-vertex	
Method	Rec.	Gen.	# Params	Accuracy	Coverage	Accuracy	Coverage	Conflict↓	Coverage	Conflict↓	Coverage
Direct Pred (8 layers)	✗	✗	27M	$40.4_{\pm 1.1}$	$13.7_{\pm 1.1}$	$13.6_{\pm 0.5}$	$1.6_{\pm 0.2}$	$179.3_{\pm 4.0}$	$19.9_{\pm 0.2}$	$198.7_{\pm 5.0}$	$6.7_{\pm 0.1}$
Direct Pred (32 layers)	✗	✗	100M	$40.2_{\pm 1.3}$	$13.6_{\pm 1.1}$	$13.1_{\pm 0.4}$	$1.6_{\pm 0.2}$	$174.0_{\pm 18.0}$	$19.1_{\pm 1.7}$	$227.7_{\pm 34.5}$	$6.5_{\pm 1.9}$
Looped TF	✓	✗	7M	$68.4_{\pm 3.7}$	$23.6_{\pm 1.9}$	$50.0_{\pm 7.6}$	$6.2_{\pm 3.2}$	$136.0_{\pm 16.1}$	$20.5_{\pm 1.5}$	$157.3_{\pm 9.0}$	$7.2_{\pm 0.7}$
HRM	✓	✗	27M	$78.7_{\pm 2.9}$	$26.7_{\pm 1.3}$	$37.4_{\pm 0.3}$	$4.7_{\pm 0.1}$	$109.7_{\pm 1.5}$	$21.8_{\pm 0.3}$	$164.3_{\pm 21.6}$	$8.9_{\pm 1.7}$
TRM	✓	✗	7M	$66.8_{\pm 5.7}$	$36.1_{\pm 22.5}$	$17.5_{\pm 11.2}$	$2.0_{\pm 1.3}$	$109.3_{\pm 3.1}$	$22.3_{\pm 0.6}$	$170.7_{\pm 17.9}$	$6.8_{\pm 0.3}$
AR	✗	✓	10.6M	$96.3_{\pm 1.0}$	$84.8_{\pm 0.8}$	$\mathbf{90.0}_{\pm 2.2}$	$53.2_{\pm 0.8}$	$19.0_{\pm 11.3}$	$83.0_{\pm 0.7}$	$61.3_{\pm 8.3}$	$40.0_{\pm 0.3}$
MDLM	✗	✓	12.6M	$96.1_{\pm 1.5}$	$87.2_{\pm 0.6}$	$74.3_{\pm 6.6}$	$47.4_{\pm 2.2}$	$\mathbf{2.7}_{\pm 0.6}$	$84.5_{\pm 4.0}$	$12.0_{\pm 7.0}$	$48.2_{\pm 1.4}$
GRAM (Ours)	✓	✓	**10M**	$\mathbf{99.7}_{\pm 0.3}$	$\mathbf{90.3}_{\pm 1.9}$	$89.7_{\pm 2.7}$	$\mathbf{57.5}_{\pm 3.4}$	$\mathbf{2.7}_{\pm 2.1}$	$\mathbf{85.8}_{\pm 0.5}$	$\mathbf{3.3}_{\pm 1.5}$	$\mathbf{51.3}_{\pm 2.8}$

cause mode collapse when multiple valid outputs exist for the same input. Additional coverage analysis is provided in Appendix D.3.

Recursive Refinement Yields Sharper Constraint Satisfaction. While generative models (AR, MDLM) achieve high coverage, GRAM consistently attains higher accuracy with comparable diversity. On N-Queens, GRAM reaches 99.7% accuracy versus 96.3% (AR) and 96.1% (MDLM). The gap is more pronounced on Graph Coloring, where GRAM reduces conflict edges to 2.7 and 3.3 on 8- and 10-vertex tasks, compared to 19.0 and 61.3 for AR. This demonstrates that recursive refinement enables stricter constraint satisfaction than generative sampling alone.

4.3 Exploring GRAM as an Unconditional Generator

Setup. To investigate GRAM's unconditional generative capability beyond conditional reasoning, we evaluate generation in two domains: structured constraint generation on *Sudoku* (from empty boards, evaluated by the fraction of generated boards satisfying Sudoku constraints) and image generation on *binarized MNIST* [15], where pixel values are thresholded to 0 or 1 (evaluated by Inception Score (IS) [42] and FID [43]). In both cases, the input is replaced by an empty conditioning signal and the model samples an output from its learned prior. Baselines include D3PM [44], a discrete diffusion model, on both tasks, and additionally a VAE [45] trained with binary reconstruction loss on MNIST. To ensure a fair comparison with existing literature, FID is calculated using real samples from the original standard MNIST.

Generative Behavior Beyond Reasoning. GRAM extends from conditional reasoning to unconditional generation in two different domains. On Sudoku generation (Figure 5), GRAM produces valid boards with 99.05% validity using 10.9M parameters and 16 supervision steps, surpassing D3PM baselines that use up to 55.1M parameters and 1000 denoising steps. Figure 7 shows qualitative examples, illustrating that the model produces diverse, fully valid boards from empty inputs without any explicit constraint checker. On MNIST (Table 2), the deterministic baseline TRM exhibits mode collapse (FID 303.29), whereas GRAM produces recognizable digits with IS and FID comparable to D3PM. Together, these results indicate that GRAM's stochastic latent

Table 2: Unconditional generation results on binarized MNIST. We report IS (↑) and FID (↓). For iterative models, a step corresponds to a supervision step for TRM and GRAM, and a denoising step for D3PM. FID is calculated using real samples with original pixel values (0–255).

Method	IS (↑)	FID (↓)
VAE	1.70	86.28
D3PM (1000 steps)	1.86	74.03
TRM (16 steps)	1.00	303.29
GRAM (Ours)		
8 steps	1.85	84.08
16 steps	1.89	77.79
32 steps	1.91	76.65
64 steps	1.95	75.39
128 steps	1.99	74.30
256 steps	**2.04**	**73.34**

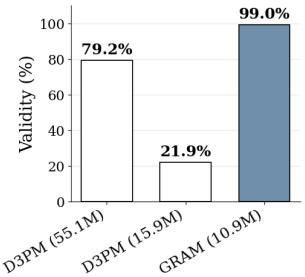
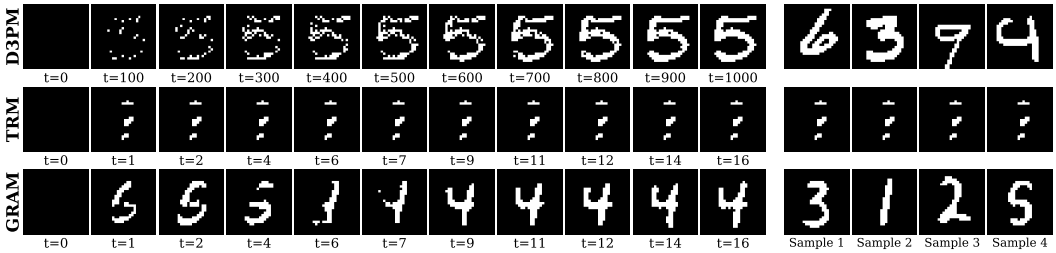


Figure 5: Unconditional Sudoku generation. Validity (%) of generated Sudoku puzzles. GRAM achieves higher validity than D3PM with substantially fewer parameters and steps.



(a) Generation Process (b) Samples

Figure 6: Visualization of the generation process and samples. (a) The generation process over recursion steps. Each row corresponds to a different model. GRAM (bottom) progressively refines the generated image through recursive latent updates, correcting initial errors. (b) Unconditional generated samples from each model.

Table 3: Ablation study on Sudoku-Extreme and N-Queens (8×8)**.** We evaluate with 5 samples. For (a), Components are added cumulatively to the Looped TF baseline (DS = deep supervision, HR = hierarchical recursion, SG = stochastic guidance). For (b), both stochasticity and learned guidance are essential—removing either significantly degrades performance.

(a) Architecture Ablation.

Model variant	Sudoku	N-Queens
base (Looped TF)	61.25	71.30
+ DS + HR (=HRM, TRM)	55.00 / 87.40	80.70 / 72.90
+ SG	65.64	86.30
+ DS + SG	73.90	100.00
+ DS + HR + SG (=GRAM)	**93.96**	**99.69**

(b) Mechanism Ablation.

Model variant	Sudoku	N-Queens
GRAM (ours)	**93.96**	**99.69**
w/o stochastic guidance	82.87	72.91
stochasticity only	94.88	50.27
guide only	0.00	0.00
w/ direct prediction	63.43	61.44
TRM w/ stochastic decoder	82.87	71.66
TRM w/ random init.	78.53	71.82

transitions support generative modeling beyond symbolic reasoning, with constraint satisfaction emerging as a natural byproduct of the recursive generative process.

Inference-Time Scaling Transfers to Generation. Table 2 further shows that increasing recursion at inference improves generation quality monotonically (IS $1.85 \rightarrow 2.04$, FID $84.08 \rightarrow 73.34$ from 8 to 256 steps), even though training uses only 16 steps. This indicates that the iterative-refinement advantage of recursive models carries over into the generative regime. Figure 6 visualizes this process; additional samples are in Section D.4.

4.4 Ablation Study

We ablate key design choices of GRAM on Sudoku-Extreme and N-Queens (8×8) using 5 samples. Table 3 summarizes the results.

Stochastic Guidance Provides Consistent Gains Across Architectures. Table 3a shows that stochastic guidance (SG) improves performance regardless of the underlying architecture: SG alone lifts the flat Looped TF baseline, and combining SG with deep supervision already reaches 100% on N-Queens. The full GRAM (with hierarchical recursion on top) achieves the best results overall (93.96% / 99.69%). While the effect of hierarchical recursion is task-dependent, SG yields consistent gains in every configuration, supporting our design of stochastic guidance as the core extension introduced by GRAM.

Both Stochasticity and Guidance Are Essential. We ablate each component by modifying the learned distribution $\epsilon_t \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2 I)$ in Equation (4). Removing guidance ($\mathcal{N}(0, \sigma_\theta^2 I)$) maintains Sudoku performance (94.88%), indicating that stochasticity alone can enable diverse reasoning paths. However, this variant collapses on N-Queens (50.27%), where structured guidance is necessary to navigate multi-solution spaces. Removing stochasticity ($\mathcal{N}(\mu_\theta, 0)$) fails completely (0.0% on both tasks), as deterministic guidance conditioned on the target leads to severe overfitting.

Naive Stochasticity Does Not Help TRM. We test two simple approaches to add stochasticity to TRM: (1) *stochastic decoding*, which samples from the output distribution instead of argmax, and

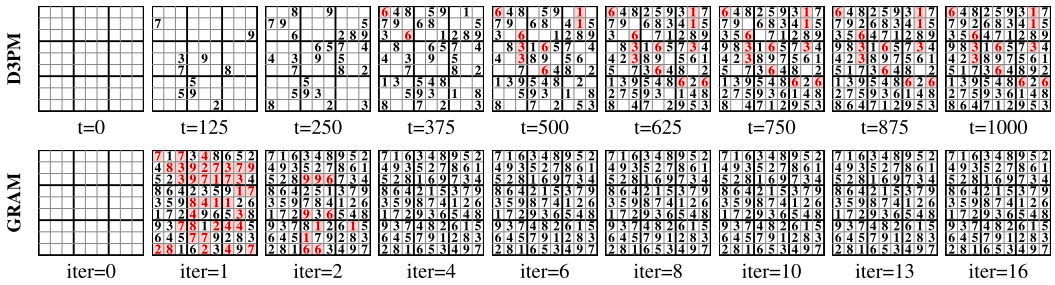


Figure 7: Qualitative examples of unconditional Sudoku generation by GRAM. Each board is independently sampled from an empty grid using the learned prior. GRAM produces diverse, complete boards satisfying all row, column, and box constraints, without an explicit constraint checker or search procedure. Incorrect digits are highlighted in red.

(2) *random initialization*, which samples z_0 from a Gaussian $\mathcal{N}(0, I)$ at each inference. Neither improves performance, demonstrating that GRAM's gains stem from the variational framework rather than mere randomness.

5 Conclusions and Limitations

We introduced GRAM, a generative framework that transforms deterministic recursive architectures into probabilistic generative models capable of modeling both $p(y \mid x)$ and $p(x)$ via recursive amortized variational inference. For reasoning problems, introducing stochasticity into latent transitions enables diverse solution discovery and improved exploration compared to deterministic counterparts. Notably, we demonstrate GRAM can leverage width-based inference-time scaling as a complement to depth: by sampling multiple latent trajectories in parallel, bypassing the latency bottleneck of depth-only scaling. Our ablations further reveal that stochastic guidance is a general-purpose extension that consistently improves any recursive architecture, and that the gains stem specifically from the variational framework — not from mere randomness, as naive stochastic alternatives applied to existing models yield no improvement.

Beyond solution-seeking, GRAM also demonstrates potential as an unconditional generative model through recursion-based generation over inputs, with generation quality improving monotonically with recursive depth even beyond training-time steps. This suggests new directions for generative modeling via hierarchical recursion. Despite these strengths, the sequential nature of deep supervision limits training efficiency compared to Transformers, posing a significant barrier to scaling GRAM toward larger foundation models.

Acknowledgment

This research was supported by the Brain Pool Plus Program (No. 2021H1D3A2A03103645) and the GRDC (Global Research Development Center) Cooperative Hub Program (RS-2024-00436165) through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (MSIT). This work was also supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00509279, Global AI Frontier Lab) and by the NYU-KAIST Global Innovation and Research Institute. Minsu Kim acknowledges funding from the KAIST Jang Young Sil Fellow Program. We are especially grateful to Gyubin and Seungju for their non-trivial contributions, and we thank the members of the MLML for valuable discussions and feedback throughout this project.

Broader Impacts

GRAM studies probabilistic recursive reasoning for structured reasoning and generation. By maintaining multiple latent trajectories, it may benefit tasks such as constraint satisfaction, and scientific problem solving, where uncertainty and multiple valid solutions are common. It also suggests a way to improve reasoning through inference-time computation rather than parameter scaling alone. Its generality also entails risks: plausible but invalid generations may be mistaken for verified solutions in downstream decision-making pipelines, and multi-sample inference may increase computational and energy costs at scale. Since our experiments focus on controlled benchmarks, deployment in real-world or high-stakes settings would require rigorous validation, uncertainty calibration, and domain-specific safeguards.

References

[1] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[2] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[3] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690, 2024.

[4] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

[5] Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. *arXiv preprint arXiv:2505.12514*, 2025.

[6] Halil Alperen Gozeten, M Emrullah Ildiz, Xuechen Zhang, Hrayr Harutyunyan, Ankit Singh Rawat, and Samet Oymak. Continuous chain of thought enables parallel exploration and reasoning. *arXiv preprint arXiv:2505.23648*, 2025.

[7] Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.

[8] Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

[9] Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

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

[11] Daniel Kahneman. Thinking, fast and slow. *Farrar, Straus and Giroux*, 2011.

[12] Yoshua Bengio. The consciousness prior. *arXiv preprint arXiv:1709.08568*, 2017.

[13] François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

[14] Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arc-agi-2: A new challenge for frontier ai reasoning systems. *arXiv preprint arXiv:2505.11831*, 2025.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[16] Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

[17] Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.

[18] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

[19] Yufan Zhuang, Liyuan Liu, Chandan Singh, Jingbo Shang, and Jianfeng Gao. Text generation beyond discrete token sampling. *arXiv preprint arXiv:2505.14827*, 2025.

[20] Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. *arXiv preprint arXiv:2505.15778*, 2025.

[21] Natasha Butt, Ariel Kwiatkowski, Ismail Labiad, Julia Kempe, and Yann Ollivier. Soft tokens, hard truths. *arXiv preprint arXiv:2509.19170*, 2025.

[22] Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation. *arXiv preprint arXiv:2502.21074*, 2025.

[23] Zhenrui Yue, Bowen Jin, Huimin Zeng, Honglei Zhuang, Zhen Qin, Jinsung Yoon, Lanyu Shang, Jiawei Han, and Dong Wang. Hybrid latent reasoning via reinforcement learning. *arXiv preprint arXiv:2505.18454*, 2025.

[24] 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. *arXiv preprint arXiv:2507.10524*, 2025.

[25] Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. Cotformer: A chain-of-thought driven architecture with budget-adaptive computation cost at inference. *arXiv preprint arXiv:2310.10845*, 2023.

[26] Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. *arXiv preprint arXiv:2410.20672*, 2024.

[27] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[29] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[30] 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.

[31] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*, 2019.

[32] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[33] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data, 2016. URL https://arxiv.org/abs/1506.02216.

[34] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers, 2016. URL https://arxiv.org/abs/1605.07571.

[35] Rahul G. Krishnan, Uri Shalit, and David Sontag. Deep kalman filters, 2015. URL https://arxiv.org/abs/1511.05121.

[36] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels, 2019. URL https://arxiv.org/abs/1811.04551.

[37] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[38] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[40] ARC-Prize-Foundation. ARC-AGI benchmarking: Leaderboard and dataset for the ARC-AGI benchmark. https://arcprize.org/leaderboard, 2026. Accessed: 2026-1-22.

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

[42] Shane Barratt and Rishi Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

[43] 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.

[44] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

[45] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[46] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[47] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[48] Simo Ryu. Minimal implementation of a d3pm (structured denoising diffusion models in discrete state-spaces), in pytorch. https://github.com/cloneofsimo/d3pm, 2024.

[49] 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.

[50] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[52] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

[53] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[54] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[55] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[56] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448, 2020.

[57] P. Erdős and A. Rényi. On the strength of connectedness of a random graph. *Acta Mathematica Academiae Scientiarum Hungarica*, 12(1):261–267, Mar 1964. ISSN 1588-2632. doi: 10.1007/BF02066689. URL https://doi.org/10.1007/BF02066689.

[58] Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luis Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 879–885. IEEE, 2019.

[59] Alexey L. Pomerantsev. Principal component analysis (pca). *Encyclopedia of Autism Spectrum Disorders*, 2014. URL https://api.semanticscholar.org/CorpusID:2534141.

[60] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, 1975. URL https://api.semanticscholar.org/CorpusID:13091446.

A Additional Method Details

A.1 Adaptive Computation Time

GRAM optionally adopts adaptive computation time (ACT) [8–10] at inference, allowing each trajectory to terminate at a learned halting depth rather than running for a fixed number of supervision steps. We follow the Q-learning formulation introduced by HRM [8] and adopted in TRM [9].

Halt head. The decoder includes an auxiliary head $q_\psi : \mathbb{R}^D \to \mathbb{R}^2$ that maps the high-level state h to two scalar values, $q_\psi(h) = (q^{\text{halt}}, q^{\text{continue}})$. These are interpreted as estimated Q-values for the binary action of halting or continuing computation at the current supervision step.

Training. The halt head is trained jointly with the main objective via a temporal-difference loss. After computing the latent state $z_T^{(n)}$ at the end of supervision step n, we form Q-learning targets:

- $\hat{q}_n^{\text{halt}} = \mathbf{1}[\hat{y}^{(n)} = y]$, indicating whether decoding the current state would yield a correct prediction.
- $\hat{q}_n^{\text{continue}} = \max\big(q_{n+1}^{\text{halt}},\, q_{n+1}^{\text{continue}}\big)$, the bootstrapped value of running one more supervision step.

The halt head is trained by regression to these targets:

$$\mathcal{L}_{\text{ACT}} = \sum_{n=1}^{N_{\text{sup}}} \Big[\big(q_n^{\text{halt}} - \hat{q}_n^{\text{halt}}\big)^2 + \big(q_n^{\text{continue}} - \hat{q}_n^{\text{continue}}\big)^2 \Big]. \tag{15}$$

This auxiliary loss is added to the main training objective and contributes only through the halt head; it does not propagate gradients into the recursive core.

Inference. At inference, computation proceeds one supervision step at a time. After each step n, we evaluate $q_\psi(h^{(n)})$ and halt if $q_n^{\text{halt}} > q_n^{\text{continue}}$, returning $\hat{y}^{(n)}$ as the prediction. Otherwise, computation continues to the next supervision step, up to a maximum budget of $N_{\text{sup}}^{\max}$ steps. Different trajectories sampled in parallel may therefore terminate at different depths, complementing the parallel-sampling scheme described in Section 2.3. In practice, we found that using only q^{halt} (halting when $\sigma(q^{\text{halt}}) > 0.5$, without the continue branch) performs comparably while simplifying implementation; our released code uses this variant.

A.2 Latent Process Reward Model (LPRM).

To rank or select among sampled candidates, we train a value head $v_\psi(z_t)$ to predict the expected accuracy of the final output, conditioned on the current latent state z_t. The LPRM is trained jointly with the main objective via a regression loss:

$$\mathcal{L}_{\text{LPRM}} = \sum_{t=1}^{T} (v_\psi(z_t) - r)^2, \tag{16}$$

where $r \in [0, 1]$ denotes the accuracy of the final prediction for a given trajectory.

A.3 Empirical Validation of the Surrogate Objective

We further analyze the approximation introduced by the surrogate training objective $\mathcal{L}_{\text{GRAM}}$ used in Section 2.2, both qualitatively and empirically.

Truncation as a gradient approximation. We frame $\mathcal{L}_{\text{GRAM}}$ as a gradient approximation rather than a separate variational objective. The full trajectory-level ELBO (Equation (13)) involves a sum of KL terms across all T_{Total} transitions, and computing its exact gradient requires backpropagation through the entire trajectory. To enable training with constant memory, we propagate gradients only through the final transition of each supervision step. This is a standard practice in recurrent latent variable models with long computation chains: ELBOs over truncated sequences are used, for example, in VRNN [33] and SRNN [34], while truncated latent imagination is used in Dreamer-family world models [37, 38]. Trading a small gradient bias for training stability via local truncation is therefore

well-precedented; what is specific to GRAM is applying this approximation at the level of recursive reasoning trajectories rather than temporal sequences.

Empirical validation. To verify that optimizing $\mathcal{L}_{\text{GRAM}}$ effectively drives improvement in the full variational bound, we compute both quantities on the validation set throughout training. The full ELBO $\mathcal{L}_{\text{ELBO}}$ is evaluated as in Equation (13), summing the reconstruction term and KL contributions across all T_{Total} transitions; the surrogate objective is evaluated as the average of $\mathcal{L}^{(n)}_{\text{GRAM}}$ over the N_{sup} supervision steps. Figure 8 reports the results on Sudoku-Extreme and N-Queens 8×8.

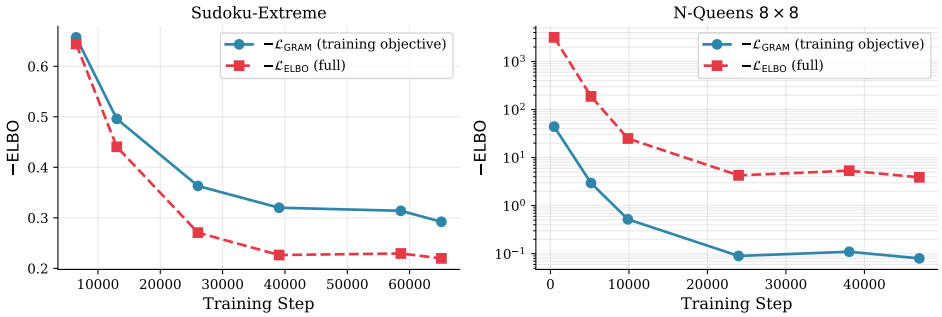


Figure 8: Full ELBO $\mathcal{L}_{\text{ELBO}}$ and surrogate objective $\mathcal{L}_{\text{GRAM}}$ throughout training (plotted as $-\text{ELBO}$, smaller is better). On both Sudoku-Extreme (left) and N-Queens 8×8 (right), both quantities decrease monotonically over training, indicating that gradient updates of $\mathcal{L}_{\text{GRAM}}$ consistently improve the full variational bound. The two curves do not coincide because $\mathcal{L}_{\text{ELBO}}$ sums KL contributions across all T_{Total} transitions while $\mathcal{L}_{\text{GRAM}}$ evaluates only the final-step KL of each supervision step; their gap reflects the cumulative KL across earlier transitions, not a failure of optimization. The N-Queens plot uses a log scale on the y-axis due to the large dynamic range.

Both $\mathcal{L}_{\text{ELBO}}$ and $\mathcal{L}_{\text{GRAM}}$ improve monotonically throughout training on both tasks. This indicates that, despite the truncation, gradient updates of $\mathcal{L}_{\text{GRAM}}$ effectively drive improvement in the full variational bound. Since $\mathcal{L}_{\text{ELBO}}$ also serves as an indirect estimate of the negative log-likelihood, its consistent improvement provides evidence that GRAM optimizes a well-defined data likelihood, even though training relies on the surrogate.

The gap between the two curves in Figure 8 reflects the structural difference between the two quantities — $\mathcal{L}_{\text{ELBO}}$ accumulates KL terms across all transitions while $\mathcal{L}_{\text{GRAM}}$ evaluates only the final-step KL of each supervision step — rather than an optimization failure. This gap is consistent with $\mathcal{L}_{\text{GRAM}}$ being a biased but useful surrogate for $\mathcal{L}_{\text{ELBO}}$.

B Training and Architecture Details

B.1 Architecture Details

GRAM consists of three components: Encoder, Recursive Core, and Decoder.

Encoder. Input tokens are mapped to embeddings via a token embedding layer, optionally concatenated with puzzle embeddings (for ARC [13, 14]), and combined with positional encodings (RoPE) [46]. The embeddings are scaled by $\sqrt{D}$ and prepended with 16 puzzle embedding tokens [8].

Recursive Core. The core maintains two latent states: h (high-level) and l (low-level). For each outer step, the low-level state is refined K times via $l \leftarrow f_L(l, h + e_x)$, injecting the input embedding at each iteration. The high-level state is then updated via $h \leftarrow f_H(h, l)$. Both f_L and f_H share the same architecture: a stack of attention and SwiGLU [47] MLP layers. In addition, as an exception, we use [SwiGLU + SwiGLU] network for the Recursive Core module instead of [Attention + SwiGLU] for Sudoku tasks, following [9]. For initialization of $z_0 = (h_0, l_0)$, we sample once from the standard Gaussian distribution $\mathcal{N}(0, I)$, then save the value within the network checkpoint and load it again, meaning the initialized z_0 has a fixed value.

Decoder. The decoder extracts content tokens from h (excluding puzzle embedding positions) and maps them to logits via a SwiGLU MLP head. An auxiliary head predicts halt decisions and correctness values from the first token of h.

Table 4: Architecture components.

Component	Module	Description
Encoder		
	Token Embedding	vocab $\to D$
	Puzzle Embedding	16 tokens (optional, for ARC)
	Position Encoding	RoPE or learned
Recursive Core		
	f_L, f_H	[Attention + SwiGLU] $\times$ 2 layers
	Iterations	K low-level, T high-level steps
	$\mu_\theta, \sigma_\theta, \mu_\phi, \sigma_\phi$	SwiGLU MLP for each parameter
Decoder		
	LM Head	Linear($D \to$ vocab)
	Q Head	Linear($D \to 2$) for halt
	V Head	Linear($D \to 1$) for value

Encoder and Decoder for Image Patches. In the MNIST [15] image generation task, we first construct a binarized dataset by normalizing the original discrete pixel values ($0 \sim 255$) to the continuous range $[0, 1]$ and applying a threshold at 0.5. For the network architecture, we employ a convolutional patch encoder, following [48, 49].

The encoding process proceeds in three stages. First, the discrete input tokens $x \in \{0, 1\}$ are normalized to the range $[-1, 1]$. Second, to capture local spatial dependencies before patchification, the normalized image passes through a shallow convolutional encoder. This encoder consists of two stacked blocks, where each block comprises a 2D convolution [50, 51] with a 5×5 kernel and padding 2, a SiLU non-linearity [52], and Group Normalization (GN) [53]. Finally, the resulting feature map is divided into non-overlapping patches of size $P \times P$ and linearly projected to match the model's hidden dimension D. The detailed architectural specifications and dimension transitions are summarized in Table 5.

Table 5: Detailed architecture of the Image Patch Encoder for MNIST. H, W denote image resolution, C input channels, P patch size, N_p the number of patches, and D the hidden dimension.

Stage	Layer / Operation	Output Dim.
1. Norm.	Input Tokens	(B, C, H, W)
	Linear Scaling $[-1, 1]$	(B, C, H, W)
2. Conv	Conv2d 5×5 ($p = 2$)	$(B, D/2, H, W)$
	SiLU $\to$ GN(32)	
	Conv2d 5×5 ($p = 2$)	$(B, D/2, H, W)$
	SiLU $\to$ GN(32)	
3. Patch	Flatten Patches	$(B, N_p, P^2 \cdot \frac{D}{2})$
	Linear Projection	(B, N_p, D)

Hyperparameters. Following Wang et al. [8], Jolicoeur-Martineau [9], both the input and output are represented as sequences of shape $[B, L]$, where B denotes the batch size and L the context length. Each input sequence includes 16 fixed puzzle embedding tokens. The latent states h_t and l_t, as well as the decoder output, have shape $[B, L, D]$, with embedding dimension D. The Transformer [39] backbone uses embedding dimension $D = 512$, attention heads N_{head}=8 , and FFN hidden dimension D_h=512. Within a recursion step, meaning a latent transition $z_t \to z_{t+1}$, we use low-level (inner) steps $K = 6$ for Sudoku [8] and $K = 4$ for all other tasks, with high-level (outer) steps $T = 3$.

B.2 Training Details

Task Configuration. All tasks represent inputs and outputs as discrete token sequences (Summarized in Table 6).

- For *Sudoku* [8], the 9×9 grid is flattened row-by-row into 81 tokens with vocabulary size 11 (0=pad, 1=blank, 2–10=digits).
- For *ARC-AGI* [13, 14], variable-size grids are padded to a fixed 30×30 canvas with EOS markers, yielding 900 tokens and vocabulary size 12 (0=pad, 1=eos, 2–11=colors); task-specific puzzle embeddings are prepended to distinguish different ARC tasks.
- *N-Queens* flattens an $N \times N$ board row-by-row into N^2 tokens with vocabulary size 3 (0=pad, 1=empty, 2=queen).
- *Graph Coloring* encodes the strict upper triangle of the adjacency matrix as $n(n-1)/2$ tokens, using 0=PAD, 1=no-edge, and 2=edge for inputs and $3 +$ color_id for output colors.
- For image generation on *MNIST* [15], images are quantized and processed via CNN-based patchification [50, 49], with the encoder applying patchify and the decoder unpatchify. Then, patched input forms 14×14 flattened sequence tokens with vocabulary size 3 (0=pad, 1=black, 2=white).

Table 6: Task-specific configurations.

Task	Seq. Len	Vocab	Puzzle Emb	Encoding
Sudoku	81	11	✗	9×9 grid, row-major
ARC-AGI	900	12	✓	30×30 padded canvas
N-Queens	N^2	3	✗	$N \times N$ board
Graph Coloring	$\frac{n(n-1)}{2}$	6	✗	Strict adjacency upper triangle
MNIST	196	3	✗	14×14 patches

Training Details. We train all models using AdamW [54] with learning rate 10^{-4}, weight decay 1.0, and gradient clipping at 1.0. The global batch size is 768. For stability, we apply exponential moving average (EMA) with decay 0.9999, following Brock et al. [55] and Song and Ermon [56]. To prevent posterior collapse, we use a KL balance [37, 38] coefficient of 0.8. The number of deep supervision steps is $N_{\text{sup}} = 16$ for all tasks. The KL coefficient β is set to 0.1 (Sudoku), $0.04/0.1$ (ARC-AGI-1/2), $0.07/0.045$ (N-Queens $8 \times 8/10 \times 10$), $0.5/0.45$ (Graph Coloring with 8/10 nodes), and 0.07 (MNIST). Task-specific training configurations are summarized in Table 7.

Table 7: Training configurations on NVIDIA RTX 4090 GPUs.

Task	Epochs	GPUs	Time
Sudoku	50K	8	2h
ARC-AGI	200K	8	5 days
N-Queens (8×8)	3K	8	1h
N-Queens (10×10)	1K	8	3h
Graph Coloring (8 nodes)	5K	8	1.5h
Graph Coloring (10 nodes)	5K	8	6h
MNIST	1.8K	8	16h

C Additional Details of Experiment Setup

C.1 Challenging Puzzle Tasks

C.1.1 Looped TF on ARC-AGI

We report Looped Transformer [7] results on Sudoku-Extreme but omit them on ARC-AGI due to prohibitive training cost. Under the same setup used for our other recursive baselines (200K epochs, batch size 768, on 8× NVIDIA RTX Pro 6000 GPUs), training Looped TF on Sudoku-Extreme already takes 19 hours, and extrapolating to ARC-AGI — which uses substantially longer sequences and a larger training set — suggests approximately **97 days** ($\approx$ 776 GPU-days) for a full training run.

This gap stems from two compounding factors. First, Looped TF lacks deep supervision: HRM, TRM, and GRAM perform N_{sup} gradient updates per trajectory (one per segment), whereas Looped TF performs only one update at the end of the full trajectory, slowing convergence. Second, Looped

TF lacks adaptive halting such as ACT [8–10], so every input must be processed for the maximum recursion depth, increasing per-example sequential compute. Both inefficiencies compound at ARC-AGI scale, making a full Looped TF training run impractical.

C.2 Multi-solution Puzzle Tasks

C.2.1 N-Queens Problem

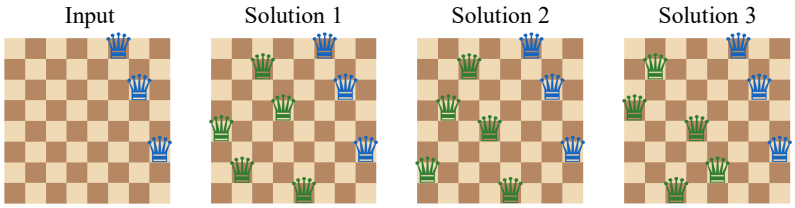


Figure 9: Example of an 8×8 N-Queens puzzle instance. In this example, 5 queens are removed from the full board, leaving 3 queens. The model must find the positions of the remaining queens. This configuration admits exactly 3 valid solutions.

Data Generation Details. The N-Queens problem requires placing N queens on an $N \times N$ chessboard such that no two queens attack each other—meaning no queens share the same row, column, or diagonal. Figure 9 illustrates an example where 5 queens are removed from an 8×8 solution, resulting in a puzzle with 3 distinct valid completions.

To construct the dataset, we first generated all valid complete N-Queens solutions for $N = 8$ and $N = 10$. We then created puzzle instances by removing a specific number of queens, treating the remaining partial configuration as the input and the original complete board as the target label. To generate instances yielding diverse valid completions, we removed $k \in \{5, 6, 7\}$ queens for the 8×8 setting and $k \in \{7, 8, 9\}$ queens for the 10×10 setting. The distribution of solution counts for our generated dataset is shown in Figure 10.

For evaluation, we employed an 85:15 train-test split. Crucially, to prevent data leakage and ensure the model learns to reason rather than memorize, the split was performed based on unique *input* configurations. This guarantees that no input pattern in the test set appears in the training set. Inputs are flattened into discrete 1D sequences $x \in \{0, 1, 2\}^L$, where $L = N^2$, along with zero-padded puzzle embedding tokens. Vocabulary mapping follows: padding (0), empty (1), and queen (2).

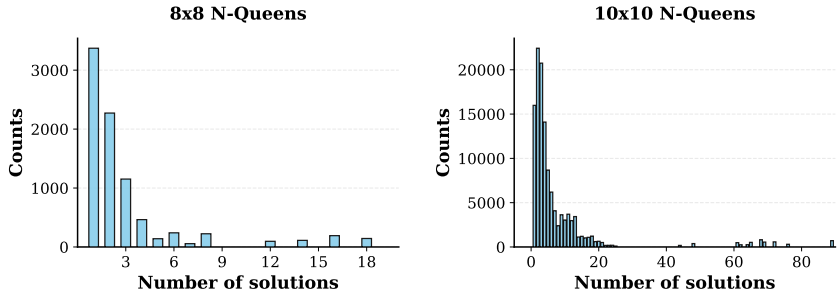


Figure 10: Distribution of the number of valid solutions for generated N-Queens instances. The dataset covers a wide range of solution counts, testing the model's ability to recover multiple valid outputs.

C.2.2 Graph Coloring Problem

Data Generation Details. The Graph Coloring problem requires assigning one of k colors to each node in a graph such that no two adjacent nodes share the same color. We consider graphs with $N \in \{8, 10\}$ nodes and use $k = 3$ colors. Figure 11 illustrates an example instance with $N = 8$ nodes and $k = 3$ colors.

Graphs are generated using the Erdős–Rényi random graph model [57], following the generation pipeline from GNN-GCP [58]. Specifically, for each instance, edges are sampled independently

with a fixed probability p, producing a symmetric adjacency matrix. We retain only graphs that are 3-colorable.

For each graph, we enumerate all valid 3-colorings and retain only canonical forms to eliminate redundant solutions under color permutation (e.g., swapping red and blue). This yields a set of structurally distinct solutions per input. The distribution of solution counts is shown in Figure 12.

The final dataset consists of 7,002 training and 255 test instances for $N = 8$, and 13,465 training and 192 test instances for $N = 10$.

Input and Output Representation. The input graph is represented by extracting the upper triangular portion of the adjacency matrix (excluding the diagonal) and flattening it into a 1D sequence. The output is a sequence of length N, where each position encodes the assigned color for the corresponding node. Vocabulary mapping is as follows: PAD (0), no edge (1), edge (2), and colors (3, 4, 5) for red, blue, and green respectively.

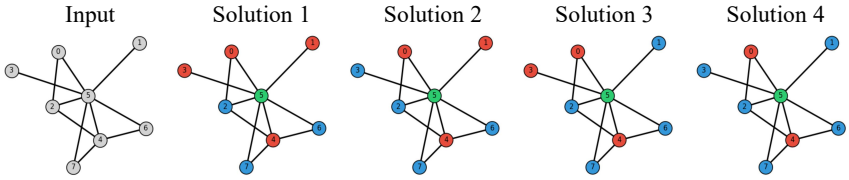


Figure 11: Graph Coloring Example

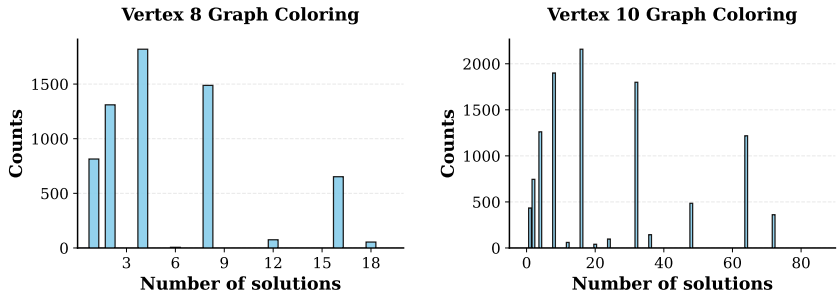


Figure 12: Distribution of the number of valid solutions for generated graph coloring instances. The dataset covers a wide range of solution counts, testing the model's ability to recover multiple valid outputs.

D Additional Experiment Results

D.1 Additional Results on Challenging Puzzle Benchmarks

Table 8 reports test accuracy on three challenging puzzle benchmarks. Here we provide additional observations complementing the main text.

GRAM Advances the Recursive-Reasoning Line. Across all three benchmarks, GRAM consistently outperforms prior recursive baselines (Looped TF, HRM, TRM) while using fewer parameters than HRM (10M vs. 27M). The complete failure of direct prediction on Sudoku and ARC-AGI-2 (0% in both cases) further confirms that recursive computation is essential for these tasks — single-pass models, regardless of capacity, cannot solve them. Together, these results indicate that GRAM's gains arise from how recursive computation is organized (probabilistic, multi-trajectory) rather than from increased model capacity.

Sudoku-Extreme Resists Parameter Scaling. All tested large reasoning models (LRMs), including Deepseek-R1 (671B), score 0% on Sudoku-Extreme. This suggests that pretrained capacity alone does not transfer to constraint-propagation reasoning, and that benchmarks like Sudoku-Extreme probe a fundamentally different axis from those captured by general-purpose LRMs. On ARC-AGI, more recent LRMs such as Gemini 3 Pro (75.0% on ARC-1, 31.1% on ARC-2) remain substantially

Table 8: Test accuracy (%) on Challenging Puzzle Benchmarks. GRAM significantly outperforms prior recursive models. All recursive model scores were obtained at 16 supervision steps.

Method	#Params	Sudoku	ARC-1	ARC-2
Large Reasoning Models				
Deepseek-R1	671B	0.0	15.8	1.3
Claude 3.7 16k	N/A	0.0	28.6	0.7
o3-mini-high	N/A	0.0	34.5	3.0
GPT 5.2 (low)	N/A	–	55.7	9.7
Grok-4-thinking	1.7T	–	66.7	16.0
Gemini 3 Pro	N/A	–	**75.0**	**31.1**
Recursive Models				
Direct Pred	27M	0.0	21.0	0.0
Looped TF	7M	61.3	-	-
HRM	27M	55.0	40.3	5.0
TRM	7M	87.4	44.6	7.8
GRAM (Ours)	**10M**	**97.0**	**52.0**	**11.1**
Human Results				
Avg. Human	–	–	60.2	–
Best Human	–	–	98.0	100.0

ahead of all recursive models, highlighting that abstract few-shot reasoning still benefits from scale; we view these numbers as benchmark-difficulty reference points rather than controlled baselines.

D.2 Scales with Parallel Sampling on ARC-AGI Challenge

To investigate the effect of GRAM's sampling on the ARC-AGI-1 benchmark, we measured performance without relying on external data augmentation. Typically, TRM achieves its reported accuracy by generating 1,000 augmentations for a single problem and performing majority voting over the results. Because this augmentation process itself creates a wide variety of samples, we isolated the specific effect of generative sampling by performing inference solely on the original problem instance and conducting majority voting over multiple sampled paths. For a fair comparison, TRM was evaluated using the same hyperparameters as GRAM, including the number of epochs, learning rate, and the number of layers.

As illustrated in Figure 13, removing augmentations causes a performance decline for both GRAM and TRM compared to the values reported in Table 8. However, in the case of GRAM, we observe that accuracy consistently improves as the model generates more parallel samples. This trend mirrors observations in Section 4.2, suggesting that increased inference-time compute through width scaling allows the model to explore more plausible reasoning trajectories and recover from initial errors, eventually leading to more robust solution discovery.

Interaction between Augmentation and Sampling. A natural question arises: why not combine higher levels of augmentation with extensive parallel sampling? To address this, we conducted an ablation study examining the interaction between data augmentation and inference-time sampling. Figure 14 presents the results across varying augmentation levels (Aug=0 to Aug=50). Without augmentation (Aug=0), increasing the number of samples yields consistent accuracy improvements, demonstrating that stochastic sampling effectively explores diverse reasoning trajectories. However, as the level of augmentation increases, the marginal benefit of additional sampling diminishes substantially. At Aug=50, performance saturates regardless of sample count—accuracy remains nearly constant whether we draw 1 or 50 samples. This observation reveals that augmentation and sampling serve complementary rather than additive roles: both mechanisms enable the model to capture solution diversity, but through different means. When training data is limited, parallel sampling compensates by exploring varied reasoning paths at inference time. When training data is abundant through augmentation, the model has already internalized sufficient diversity during training, rendering additional inference-time exploration redundant. Consequently, scaling sampling beyond augmentation provides diminishing returns, justifying our experimental design choice to evaluate these two scaling axes separately.

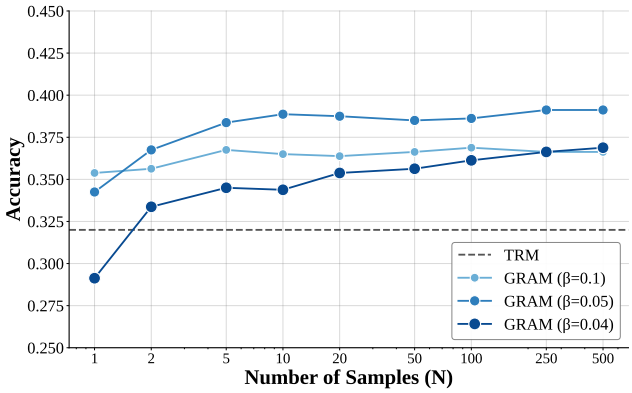


Figure 13: Effect of sampling on ARC-AGI-1 without data augmentation. To isolate the internal sampling effect, both models are evaluated on original problem instances without 1,000 augmentations. While removing augmentations causes an initial performance drop, GRAM exhibits robust scaling through generative sampling as the number of parallel samples N increases, outperforming the TRM baseline.

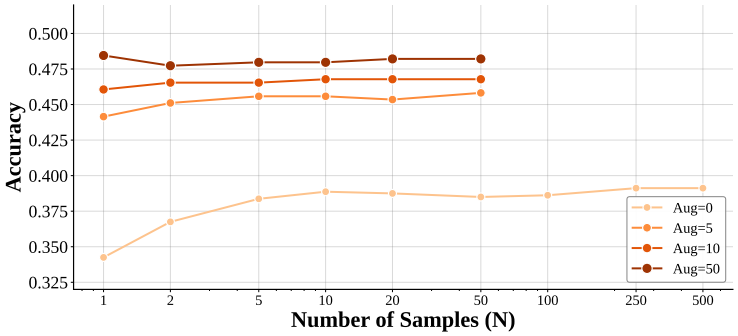


Figure 14: Effect of augmentation on sampling efficiency. With limited augmentation (Aug=0), parallel sampling provides consistent gains. As augmentation increases, sampling benefits diminish—at Aug$= 50$, performance saturates regardless of sample count, suggesting augmentation and sampling serve complementary roles in capturing solution diversity.

D.3 Solution Coverage Analysis

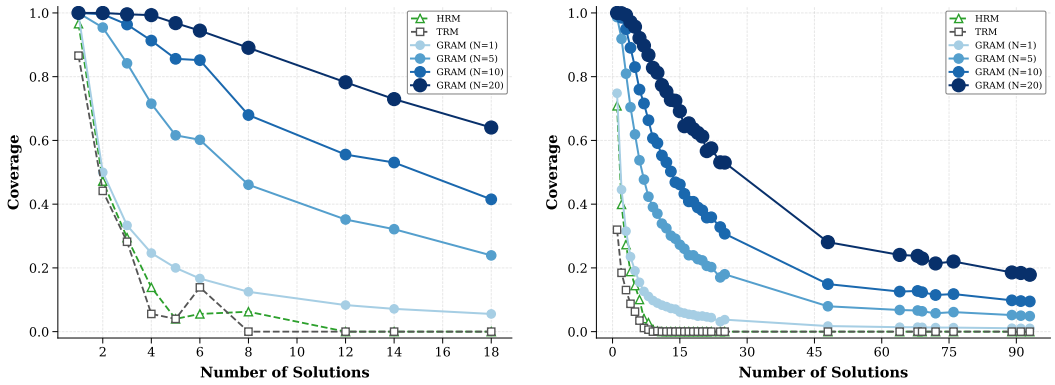
We analyze the ability of GRAM to capture the diversity of the solution space compared to deterministic baselines. Figure 15 presents the solution coverage on 8×8 and 10×10 N-Queens tasks with respect to the total number of valid ground-truth solutions.

As shown in Figure 15, deterministic recursive models (HRM and TRM) exhibit a sharp decline in coverage as the number of possible solutions increases. Since these models are constrained to a single fixed reasoning trajectory, they structurally fail to explore alternative paths, resulting in severe mode collapse in multi-solution landscapes.

In contrast, GRAM effectively leverages its generative latent transitions to cover a broader range of solutions. As the number of parallel samples N increases (from 1 to 20), the solution coverage improves monotonically across both 8×8 and 10×10 settings. This empirical evidence confirms that GRAM's stochastic guidance mechanism is essential for navigating complex problem spaces where multiple valid reasoning paths exist.

D.4 Additional Generated Image Samples

In this section, we provide further qualitative results demonstrating GRAM's capability in unconditional image generation. Figure 16 presents a diverse set of samples generated on the binarized MNIST dataset, visualized across the recursive inference steps $t = 0$ to $t = 16$.



(a) N-Queens 8×8 **(b)** N-Queens 10×10

Figure 15: Solution coverage analysis on N-Queens (8×8 and 10×10) with respect to the number of ground-truth solutions. While deterministic baselines (HRM, TRM) suffer from mode collapse as the solution space grows, GRAM demonstrates monotonic improvement in coverage as the number of parallel samples N increases.

As observed in the main text, GRAM exhibits a distinct progressive refinement behavior. Starting from a black initialization, the model iteratively adds details and sharpens the structure of the digit. A particularly compelling property of this process is the model's ability to recover from initially ambiguous or incorrect formations.

For instance, in the second row (generating the digit '2') and the last row (generating the digit '1'), the early predictions at $t = 1$ and $t = 2$ manifest as disjointed artifacts or incorrect shapes. However, as the recursion proceeds, GRAM effectively leverages its feedback loop to correct these initial errors, resolving the ambiguity and converging to a coherent, high-quality digit by $t = 16$.

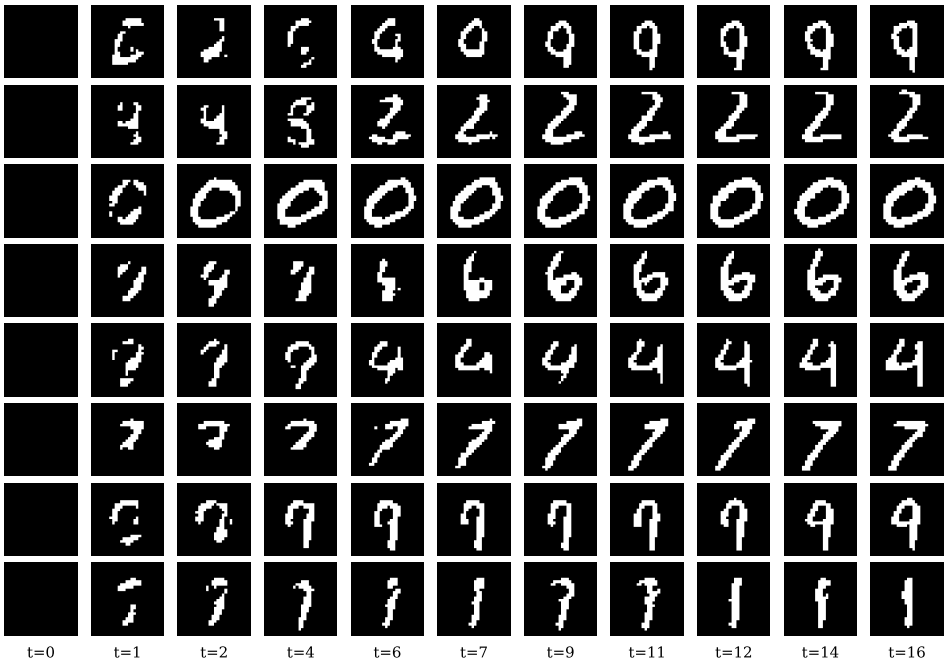


Figure 16: Additional generated samples from GRAM. We provide 8 additional samples generated unconditionally on binarized MNIST using GRAM. Each row represents a single generated sample, visualized across its recursive refinement process.

D.5 Additional Experiment Results on Unconditional Sudoku Generation

In this section, we provide additional details on unconditional Sudoku generation. Unlike the conditional Sudoku-solving setting, where the input board contains given clues, the model receives an entirely blank board and samples a complete 9×9 Sudoku board from its learned prior. We evaluate each generated board using the standard Sudoku validity criterion: every row, column, and 3×3 box must contain the digits 1 through 9 exactly once. We report the validity rate over 100K generated boards. To check whether high validity comes from repeatedly producing the same board, we also compute the fraction of unique boards among valid samples.

For GRAM, we construct the unconditional training set from Sudoku-Extreme [8], the Sudoku benchmark used by HRM and TRM. We sample 50K complete solutions from the original training split, discard the clue patterns, and use an all-blank board as input with the complete solution as the target. No data augmentation is used. We train GRAM on this derived 50K-solution set for 200 epochs with learning rate 10^{-4}, EMA decay 0.999, and KL coefficient 0.05. The resulting model contains 10.9M parameters and uses 16 inference steps.

For D3PM baselines, we use a DiT-style Transformer backbone and evaluate two model sizes. D3PM-Big uses hidden dimension 768, 5 Transformer blocks, and 12 attention heads, yielding 55.1M parameters, while D3PM-Small uses hidden dimension 512, 3 Transformer blocks, and 8 attention heads, yielding 15.9M parameters. Both variants are trained on the same derived training set and generate boards with 1000 denoising steps.

As shown in Table 9, GRAM achieves 99.05% validity, outperforming all D3PM baselines. The strongest D3PM baseline, D3PM-Uniform (Big), reaches 91.33% validity while using 55.1M parameters and 1000 denoising steps. In contrast, GRAM uses fewer parameters and only 16 inference steps. In all cases, the valid samples are unique under exact board matching, indicating that the reported validity is not due to simple repetition of a small set of boards. These results show that GRAM can generate highly constrained symbolic structures from an empty input, supporting its potential as a generator beyond conditional puzzle solving.

Figure 17 illustrates the unconditional Sudoku generation setup. Starting from an empty board, the task is to generate complete boards, and validity is determined by whether the generated board satisfies all Sudoku constraints. Figure 7 shows qualitative examples of boards generated by GRAM.

Table 9: Unconditional Sudoku generation. We report the ratio of generated boards satisfying Sudoku constraints over 100K samples. All valid boards are unique for all methods in this evaluation.

Method	#Params	Steps	Validity(%)
D3PM-Uniform (Big)	55.1M	1000	91.33
D3PM-Uniform (Small)	15.9M	1000	29.24
D3PM-Absorb (Big)	55.1M	1000	79.18
D3PM-Absorb (Small)	15.9M	1000	21.88
GRAM (Ours)	**10.9M**	**16**	**99.05**

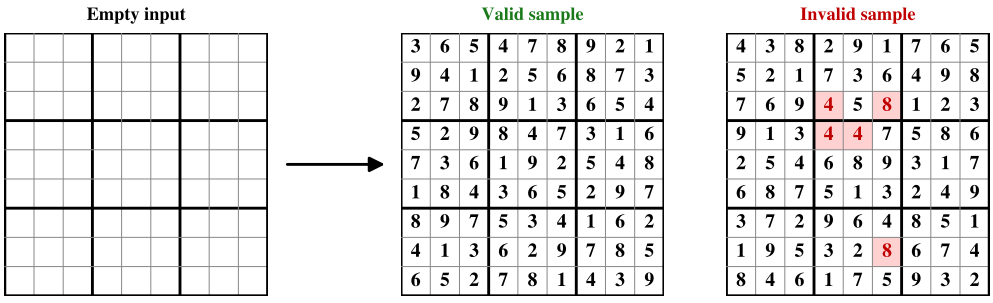


Figure 17: Unconditional Sudoku generation setup. Starting from an empty board, the task is to generate complete Sudoku boards. The valid sample satisfies all Sudoku constraints, while red entries in the invalid sample indicate cells involved in constraint violations.

D.6 Visualizing Latent Recursion Process

To understand how stochastic guidance shapes reasoning, we visualize latent trajectories during recursive computation. Specifically, we track the high-level state h at each supervision step throughout the recursion process. For visualization, we project these latent vectors into 2D using PCA [59] and interpolate unobserved states via K-D tree [60] to construct a continuous loss landscape.

Figures 18 and 19 compare TRM and GRAM on the same Sudoku puzzle. TRM follows a single deterministic path from initialization to solution, offering no mechanism to escape if the trajectory enters a suboptimal region. In contrast, GRAM samples diverse trajectories that explore different regions of latent space before converging. While some trajectories become trapped in local minima (bright yellow regions), others successfully navigate toward the global optimum (dark blue regions). This diversity enables GRAM to discover valid solutions more reliably through parallel exploration.

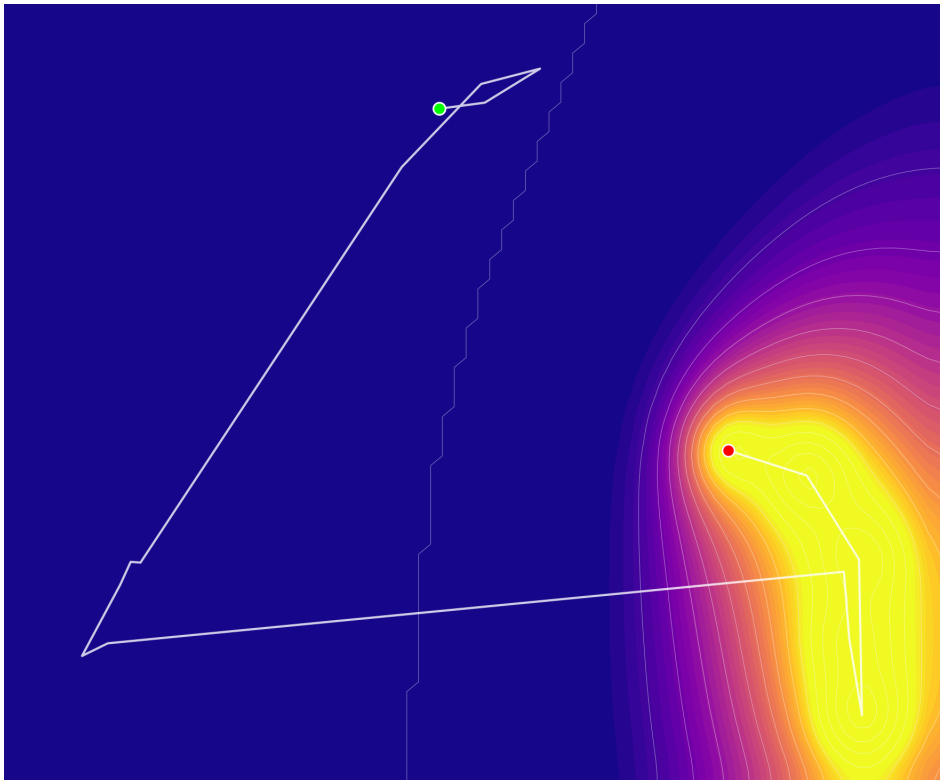


Figure 18: Latent reasoning trajectory of TRM. The red dot indicates the initial state h_0 and the green dot indicates the final state h_T. Background color represents the loss landscape: bright yellow corresponds to high loss regions, while dark blue indicates low loss (optimal) regions. TRM follows a single deterministic path with no ability to escape suboptimal trajectories.

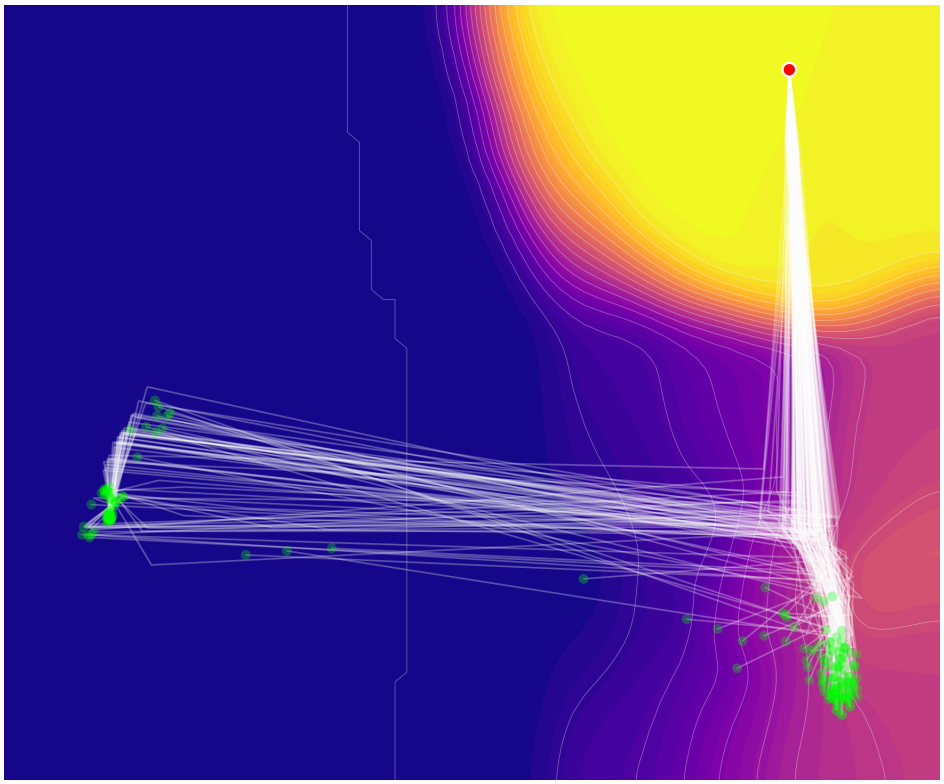


Figure 19: Latent reasoning trajectories of GRAM (50 samples). Using the same visualization scheme as Figure 18, we show 50 sampled trajectories from GRAM. The stochastic guidance enables diverse exploration of the latent space: while some trajectories converge to local minima (right bottom), others successfully reach the global optimum (left middle), demonstrating how parallel sampling improves solution discovery.

Licenses

Table 10: Existing assets, licenses, and source links. We list the existing datasets, benchmarks, and public reference implementations used or cited in our experiments. Synthetic N-Queens and Graph Coloring instances are generated by the authors and are therefore not external assets.

Asset	Use in this paper	License / terms	Source link
MNIST	Binarized MNIST generation experiments	Creative Commons Attribution-Share Alike 3.0	https://keras.io/api/datasets/mnist/
ARC-AGI-1 / original ARC	ARC-AGI reasoning benchmark	Apache License 2.0	https://github.com/fchollet/ARC-AGI
ARC-AGI-2	ARC-AGI-2 reasoning benchmark / reference results	Apache License 2.0	https://github.com/arcprize/ARC-AGI-2
HRM repository	HRM baseline and Sudoku-Extreme-related reference implementation	Apache License 2.0	https://github.com/sapientinc/HRM
TinyRecursiveModels / TRM repository	TRM baseline and recursive reasoning reference implementation	MIT License	https://github.com/SamsungSAILMontreal/TinyRecursiveModels
MDLM repository	Masked diffusion baseline reference implementation, if public code is used	Apache License 2.0	https://github.com/kuleshov-group/mdlm
Google Research D3PM implementation	D3PM image-generation baseline reference implementation, if public code is used	Apache License 2.0	https://github.com/google-research/google-research/blob/master/d3pm/images/diffusion_categorical.py
Looped Transformer repository	Looped Transformer baseline reference implementation, if public code is used	MIT License	https://github.com/Leiay/looped_transformer
N-Queens	Synthetic multi-solution constraint satisfaction task generated by the authors	Not an external asset	N/A
Graph Coloring	Synthetic multi-solution constraint satisfaction task generated by the authors	Not an external asset	N/A