GENERATIVE RECURSIVE REASONING MODELS

Junyeob Baek[1*], **Mingyu Jo**[1*], **Minsu Kim**[1,2], **Yoshua Bengio**[2,3], **Sungjin Ahn**[1,4]
[1]KAIST, [2]Mila – Québec AI Institute, [3]Université de Montréal, [4]NYU

ABSTRACT

We introduce Generative Recursive reAsoning Models (GRAM), a recursion-based generative model that is effective for complex planning and reasoning problems. GRAM reformulates recent latent recursive architectures as a stochastic generative process with probabilistic latent transitions, enabling efficient and stable computation entirely in latent space without relying on token-level sequences as in chain-of-thought (CoT) prompting. We optimize this generative recursion via amortized variational inference, allowing the model to represent and explore multiple plausible latent trajectories conditioned on the input. This formulation supports both conditional reasoning through $p(y \mid x)$ and unconditional generative modeling through $p(x)$. Empirically, GRAM achieves strong performance on challenging reasoning benchmarks, competitive with much larger language models on ARC-1 and ARC-2, demonstrating the effectiveness of recursion-based generative modeling for System 2 tasks.

1 INTRODUCTION

Solving complex problems such as mathematics and abstract puzzles has become a central benchmark for evaluating advanced reasoning in artificial intelligence (Gao et al., 2024; Sun et al., 2025; Chollet, 2019; Chollet et al., 2025). While most of modern deep learning excels at *System 1* intelligence—fast, pattern-based inference via large-scale pretraining—these tasks require *System 2* reasoning, characterized by long-horizon planning, iterative refinement, and algorithmic search (Kahneman, 2011; Bengio et al., 2019).

In large language models (LLMs), System 2 reasoning is typically approximated by externalizing intermediate steps as token-level sequences, mostly through reinforcement learning (Lightman et al., 2023; Guo et al., 2025) with Chain-of-Thought (CoT) prompting (Wei et al., 2022; Yao et al., 2023). Although effective, this paradigm suffers from several limitations. It relies on brittle step decompositions, is vulnerable to cascading errors, and incurs high computational cost due to long autoregressive decoding chains (Zhang et al., 2022; Tyen et al., 2024; Wang et al., 2025). More fundamentally, it treats reasoning as a linguistic artifact rather than as an internal computational process.

In contrast, reasoning need not be represented as explicit token-level sequences (Hao et al., 2024). Representing intermediate steps in a continuous latent space allows iterative computation without committing to brittle symbolic decompositions or long autoregressive chains (Zhu et al., 2025; Zhang et al., 2025). From a computational perspective, latent-state updates enable compact representations, efficient refinement, and stable recurrent computation, directly addressing the structural limitations of token-based reasoning in LLMs (Zhang et al., 2025; Zhuang et al., 2025; Geiping et al., 2025).

Notably, human reasoning also appears to operate largely in such an internal and latent manner, progressing through iterative updates to abstract cognitive states rather than overt linguistic expressions (Coetzee et al., 2022; Mahowald et al., 2024). This perspective motivates latent-space reasoning models that perform iterative computation directly in continuous state spaces. Recent recursive latent models, including Hierarchical Reasoning Models (HRM) (Wang et al., 2025) and Tiny Recursive Models (TRM) (Jolicoeur-Martineau, 2025), revisit this paradigm with modern training techniques that enable stable and memory-efficient deep recurrence.

[*]Equal contribution

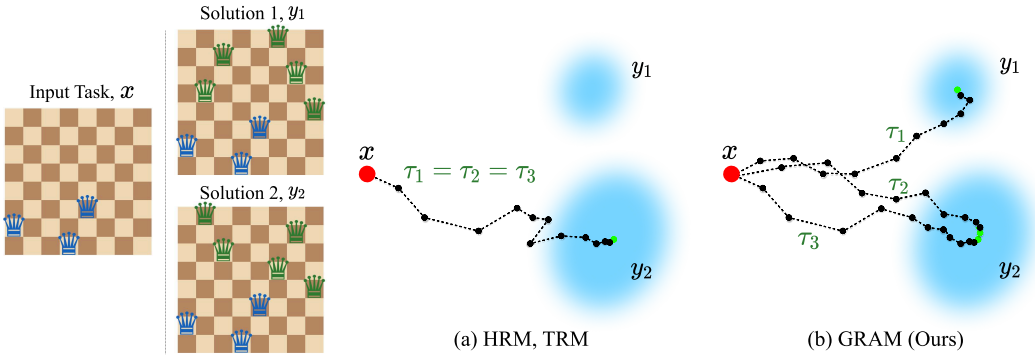


Figure 1: **Comparison of Latent Reasoning Trajectories.** *Left*: An N-Queens problem with two valid solutions y_1 and y_2. *Right*: Given three independent inference runs (τ_1, τ_2, τ_3) from the same input x: (a) Prior recursive models (HRM, TRM) are deterministic—all runs collapse to an identical trajectory, converging to a single solution and failing to explore alternatives. (b) GRAM injects stochastic guidance at each recursion step, producing diverse trajectories that reach multiple valid solutions y_1 and y_2, while naturally enabling parallel inference-time scaling.

In particular, HRM introduces a dual-loop hierarchical multi-scale architecture that separates reasoning into a high-frequency, low-level module for detailed computation and a low-frequency, high-level module for abstract planning. By combining this structured recurrence with *one-step gradient approximation* and *deep supervision* (Wang et al., 2025), these models achieve effective $\mathcal{O}(1)$ training cost per recursion step and demonstrate that compact recurrent architectures can match or exceed LLMs on challenging benchmarks such as ARC-AGI (Chollet, 2019; Chollet et al., 2025).

Despite their success, existing recursive reasoning models are fundamentally *deterministic*: given an input, they converge to a single latent trajectory and a single solution. This structural property limits their ability to represent uncertainty, to explore alternative hypotheses, and to solve problems with multiple valid solutions. More generally, deterministic recursion induces a single attractor in latent space, effectively collapsing the distribution of plausible reasoning paths into one fixed point.

In this work, we propose *Generative Recursive reAsoning Models (GRAM)*, a framework that reformulates recursive latent reasoning as probabilistic inference over distributions of reasoning trajectories. Unlike prior recursive models that compute a single deterministic refinement path, GRAM treats reasoning as a stochastic generative process, $z_{t+1} \sim p_\theta(z_{t+1} \mid z_t, x)$, thereby defining a full conditional generative model $p_\theta(y \mid x)$ over solutions given inputs. By fixing the input to a constant token, this formulation naturally extends to an unconditional model $p_\theta(x)$, enabling GRAM to learn not only to solve problems but also to model the input distribution itself.

This formulation yields three key advantages. First, while deterministic recursion induces a single attractor in latent space and collapses plausible reasoning programs into one fixed trajectory, GRAM represents a distribution over reasoning trajectories, enabling the discovery of multiple valid solutions and reasoning under ambiguity (see Figure 1). Second, stochastic latent transitions introduce a new axis of inference-time scaling, termed *width scaling*, which improves solution quality through parallel exploration without increasing model size or sequential depth. Third, by formulating a probabilistic framework capable of modeling both the conditional solution distribution $p_\theta(y|x)$ and the unconditional input distribution $p_\theta(x)$, GRAM supports unsupervised learning, enabling recursive reasoning models to learn from unlabeled data in addition to supervised problem–solution pairs.

Empirically, GRAM achieves strong performance on challenging benchmarks such as Sudoku-Extreme (Wang et al., 2025) and ARC-AGI (Chollet, 2019; Chollet et al., 2025), consistently outperforming deterministic recursive baselines. On tasks with multiple valid solutions, including constraint satisfaction problems, GRAM succeeds where deterministic models structurally fail and outperforms autoregressive and diffusion baselines. We further show that the framework generalizes beyond symbolic reasoning to unconditional image generation on MNIST (Lecun et al., 1998), where GRAM achieves comparable IS and FID to D3PM (Austin et al., 2021), with quality improving monotonically over additional recursive refinement steps.

2 GENERATIVE RECURSIVE REASONING MODELS

In this section, we introduce **Generative Recursive reAsoning Models (GRAM)**, a recursive architecture designed to solve complex reasoning tasks such as ARC-AGI (Chollet, 2019; Chollet et al., 2025) and Sudoku (Wang et al., 2025) with orders of magnitude fewer parameters than LLMs. GRAM builds on latent-state reasoning and introduces a novel generative formulation that enhances its ability to search and generalize. We first describe the architecture of GRAM in Section 2.1, followed by the training procedure in Section 2.2, and test-time inference in Section B.1. An overview of the GRAM architecture is shown in Figure 2.

2.1 ARCHITECTURE OF GRAM

Through its generative formulation, GRAM can broadly define a distribution over outputs—either conditioned on an input $p_\theta(y|x)$ for reasoning tasks, or unconditionally $p_\theta(x)$ for generation tasks. We primarily focus on the conditional setting throughout this section.

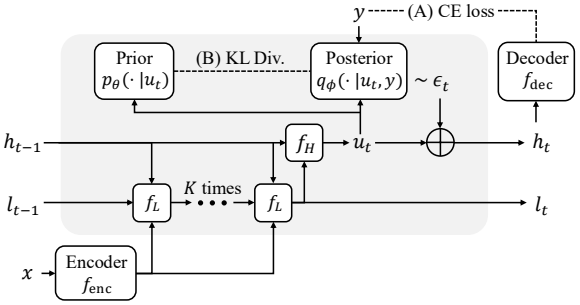


Figure 2: **GRAM Architecture Overview.** A single recursion step from $t-1$ to t. After K low-level refinements via f_{L}, the high-level update f_{H} produces u_t, to which stochastic guidance ϵ_t is added: $h_t = u_t + \epsilon_t$.

Abstract Architecture. GRAM is a recurrent latent reasoning model that incrementally refines a hidden solution trajectory $z_0 \to \cdots \to z_T$, where each latent state $z_t = (h_t, l_t)$ consists of a high-level component h_t (representing abstract reasoning anchor) and a low-level component l_t (representing iterative local computations).

Given an input problem x, the model with parameter θ operates as follows:

$$e_x = f_{\text{enc}}(x;\theta), \quad z_0 = \text{fixed init} \tag{1}$$

$$z_t \sim p_\theta(\cdot|z_{t-1}, e_x), \quad t = 1 \dots T \tag{2}$$

$$\hat{y} = \arg\max f_{\text{dec}}(h_T;\theta). \tag{3}$$

Here, f_{enc} encodes the input into embedding e_x, and f_{dec} maps the final high-level state h_T to a predicted output $\hat{y}$. We now describe the latent transition distribution $p_\theta(\cdot|z_{t-1}, e_x)$ in detail.

Hierarchical and Generative Latent Transitions. Unlike classical RNNs that operate over a single deterministic hidden state and often suffer from premature convergence (Wang et al., 2025), GRAM maintains a structured latent state $z = (h, l)$ with two interacting levels: a high-level state h capturing abstract reasoning plans, and a low-level state l encoding fine-grained computations.

Inspired by HRM and TRM, GRAM adopts a multiscale update scheme. At each outer step t, the low-level state is refined over K inner steps while the high-level state remains fixed:

$$l_{t,k} = f_{\text{L}}(h_{t-1}, l_{t,k-1}, e_x;\theta), \quad k = 1, \dots, K, \tag{4}$$

with initialization $l_{t,0} := l_{t-1}$. After K low-level updates, the high-level state is updated via a stochastic transition conditioned on the refined low-level state:

$$u_t = f_{\text{H}}(h_{t-1}, l_t;\theta), \tag{5}$$

$$\epsilon_t \sim p_\theta(\epsilon_t \mid u_t) := \mathcal{N}\big(\mu_\theta(u_t), \sigma_\theta^2(u_t) I\big), \tag{6}$$

$$h_t = u_t + \epsilon_t. \tag{7}$$

Here, we define $l_t := l_{t,K}$, completing the transition from $z_{t-1} = (h_{t-1}, l_{t-1})$ to $z_t = (h_t, l_t)$ via the stochastic latent transition distribution

$$z_t \sim p_\theta(\cdot \mid z_{t-1}, x) = p_\theta(\cdot \mid z_{t-1}, e_x), \tag{8}$$

where p_θ encapsulates the coupled low-level refinement (Equation (4)) and high-level stochastic update (Equations (5) to (7)).

Learnable Stochastic Guidance. The injected noise ϵ_t serves as a learnable stochastic guidance signal that enables the model to explore diverse yet directed reasoning paths. Instead of adding unstructured noise, GRAM learns a structured distribution conditioned on the current high-level state:

$$\epsilon_t \sim p_\theta(\epsilon_t \mid u_t) = \mathcal{N}(\mu_\theta(u_t), \sigma_\theta^2(u_t)I). \tag{9}$$

The mean $\mu_\theta(u_t)$ encodes a preferred direction of reasoning progression, while the variance $\sigma_\theta^2(u_t)$ controls the degree of stochastic exploration. This design allows GRAM to capture uncertainty, prevent premature convergence, and support robust exploration of the solution space. Unlike deterministic hierarchical models such as HRM or TRM, which commit to a single latent trajectory, GRAM enables parallel sampling and diversity through generative recursion in latent space.

Architectural Instantiation. Both f_{L} and f_{H} are instantiated as shared 2-layer Transformer networks (Vaswani et al., 2017) with RMSNorm (Zhang & Sennrich, 2019), rotary positional embeddings (RoPE) (Su et al., 2024), and SwiGLU activations (Shazeer, 2020). The stochastic guidance mechanism is implemented via an MLP with SwiGLU activations that parameterizes both posterior μ_ϕ, σ_ϕ, and prior μ_θ, σ_θ parameters. The encoder f_{enc} and decoder f_{dec} are task-specific and vary depending on the target domain. Additional architectural details are provided in Appendix D.1.

2.2 Training of GRAM

GRAM is trained to model the conditional distribution $p(y \mid x)$, 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.

To enable efficient training over long recursive latent trajectories, we apply a gradient approximation together with deep supervision, enabling constant-time backpropagation.

Latent Variable Modeling. We model GRAM as a latent-variable probabilistic model p_θ, where the generative trajectory $\tau = (z_0 \to \cdots \to z_T)$ consists of a sequence of latent variables. 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:

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

Our training objective is to maximize the ELBO over the training dataset.

In practice, the conditional log-likelihood $\log p_\theta(y \mid \tau, x)$ is computed via a log-softmax over the decoder f_{dec} applied to the terminal latent state. During training, latent trajectories are sampled from the variational posterior $q_\phi(\cdot \mid x, y)$, while the learned prior $p_\theta(\cdot \mid x)$ is used at inference time.

The prior over latent trajectories is modeled as a Markov process,

$$p_\theta(\tau \mid x) = p(z_0) \prod_{t=1}^{T} p_\theta(z_t \mid z_{t-1}, x), \tag{12}$$

where z_0 denotes a fixed initial state shared by both the prior and the posterior.

Similarly, the variational posterior $q_\phi(\tau \mid x, y)$ is defined with the same Markov transition structure as the prior, but parameterized independently by ϕ. In particular, posterior transitions are implemented via a reparameterized Gaussian noise (similar to Equation (6)), mirroring the prior dynamics while conditioning on the target output y.

Given that the prior and posterior share the same Markov structure and all stochasticity in the latent trajectory is introduced via noise variables $\epsilon_{1:T}$, we can equivalently express the trajectory

distributions $p_\theta(\tau \mid x)$ and $q_\phi(\tau \mid x, y)$ as distributions over $\epsilon_{1:T}$. Consequently, the trajectory-level KL decomposes into a sum of step-wise KL divergences:

$$\mathrm{KL}(q_\phi(\tau \mid x, y) \,\|\, p_\theta(\tau \mid x)) = \sum_{t=1}^{T} \mathbb{E}_{q_\phi(\epsilon_{<t} \mid x, y)} \Big[\mathrm{KL}(q_\phi(\epsilon_t \mid u_t, y) \,\|\, p_\theta(\epsilon_t \mid u_t)) \Big].$$

Here, $u_t = f_{\mathrm{H}}(h_{t-1}, l_t)$ denotes the deterministic high-level update before noise injection, as defined in Equation (7). 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.

Truncated ELBO and Gradient Approximation. Computing exact gradients of $\mathcal{L}_{\mathrm{ELBO}}$ requires backpropagation through all T recurrent transitions, incurring $\mathcal{O}(T)$ computational and memory cost. Following prior work on hierarchical and recursive reasoning models (Wang et al., 2025; Jolicoeur-Martineau, 2025), we adopt a gradient approximation by truncating gradient flow through the latent trajectory. Specifically, following Jolicoeur-Martineau (2025), we detach intermediate states $h_{<T}$ from the computational graph and propagate gradients only through the final transition step, along with the encoder and decoder.

This yields the following truncated ELBO objective,

$$\tilde{\mathcal{L}}_{\mathrm{ELBO}}(x, y; \theta, \phi) = \mathbb{E}_{q_\phi} \big[\log p_\theta(y \mid h_T, x) \big] - \mathrm{KL}\big(q_\phi(\epsilon_T \mid u_T, y) \,\|\, p_\theta(\epsilon_T \mid u_T)\big), \tag{13}$$

where $h_T = u_T + \epsilon_T$ and gradients are stopped for $z_{<T}$. Importantly, $\tilde{\mathcal{L}}_{\mathrm{ELBO}}$ should be interpreted as a biased yet computationally efficient gradient estimator of the full ELBO, rather than a different variational objective. Detailed derivation of truncated ELBO is provided in Appendix C.1.

Overall Training Scheme with Deep Supervision. The truncated ELBO described above optimizes a single segment of T recursion steps, called a supervision step. To provide dense learning signals across the full inference trajectory, we apply deep supervision by repeating this process over N_{sup} consecutive supervision steps (Wang et al., 2025; Jolicoeur-Martineau, 2025).

After each supervision step, the model (1) computes the truncated ELBO, (2) updates parameters, and (3) carries over the detached final state $\mathrm{stopgrad}(z_T)$ as the initial state z_0 for the next supervision step. This sequential process allows the model to refine its latent state across supervision steps while maintaining constant memory cost, as gradients do not propagate across step boundaries.

Overall, GRAM is trained by jointly optimizing θ and ϕ to maximize the truncated ELBO $\tilde{\mathcal{L}}_{\mathrm{ELBO}}(x, y; \theta, \phi)$ over the training dataset. Detailed training procedures and hyperparameters are provided in Appendix D.3.

Inference-Time Scaling. With the nature of deep supervision objectives, it supports flexible recursion length at inference time, regardless of the used recursion length at training time. We provide further method details in Appendix B.1.

3 Experiment

In this section, we conduct a series of experiments to empirically evaluate and analyze the effectiveness of GRAM. We first evaluate the reasoning performance on challenging puzzles, specifically Sudoku and ARC-AGI (Section 3.1). We then assess the model's ability to capture diverse solution paths in puzzles with multiple solutions (Section 3.2), and extend this approach to unconditional image generation tasks (Section 3.3). Finally, we perform an ablation study to evaluate the impact of key design choices (Section F.5).

3.1 Challenging Puzzle Tasks

Setup. We evaluate on Sudoku-Extreme (Wang et al., 2025), which contains 9×9 puzzles with minimal clues requiring extensive constraint propagation, and ARC-AGI Challenge (Chollet, 2019; Chollet et al., 2025), which tests abstract visual reasoning through few-shot pattern recognition. We compare against direct prediction (Transformer (Vaswani et al., 2017)), recursive baselines (HRM (Wang et al., 2025), TRM (Jolicoeur-Martineau, 2025)), and state-of-the-art Large reasoning models (ARC-Prize-Foundation, 2026). For the scaling analysis, we reproduce TRM following Jolicoeur-Martineau (2025) under identical settings for a fair comparison.

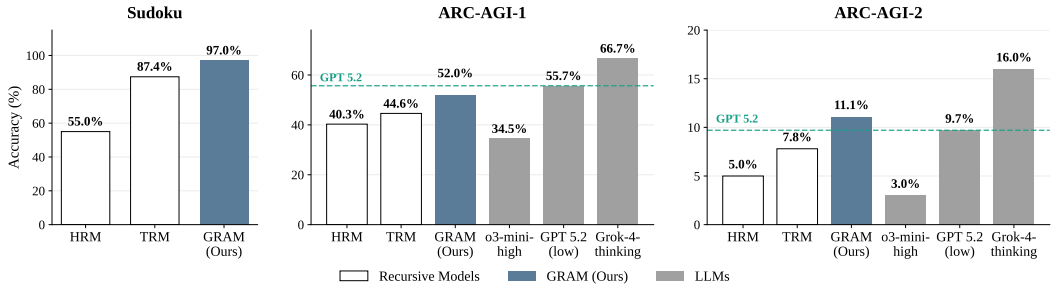


Figure 3: **Performance on puzzle benchmarks.** We compare recursive models (white), GRAM (blue), and large reasoning models (gray). On Sudoku-Extreme, GRAM achieves 97.0%, outperforming TRM (87.4%) and HRM (55.0%). On ARC-AGI, GRAM surpasses GPT-5.2 level (green dashed) on ARC-2 (11.1% vs 9.7%) and approaches it on ARC-1 (52.0% vs 55.7%), despite using 10M parameters—orders of magnitude smaller than LLMs.

Stochastic Guidance Improves Reasoning. Figure 3 summarize our main results. GRAM consistently outperforms prior recursive models across all benchmarks and notably achieves performance comparable to GPT-5.2 on ARC-AGI. We attribute this improvement to the fundamental difference in how reasoning trajectories are utilized. While 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 sort-of-art methods, are provided in Appendix 3.1.

Parallel Sampling Provides a New Test-time Scaling Axis. Figure 4 shows that increasing the number of parallel samples consistently improves performance across all iteration counts. Notably, GRAM with $N = 20$ samples at 16 iterations significantly outperforms TRM at 320 iterations (97.0% vs 90.5%), despite requiring the same computational budget. While existing recursive models scale only through sequential refinement—an inherently slow process—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 B.1). This parallel scaling bypasses the latency bottlenecks of depth-based scaling while achieving superior performance. Additional analysis evaluated on ARC-AGI Challenge is provided in Appendix F.2.

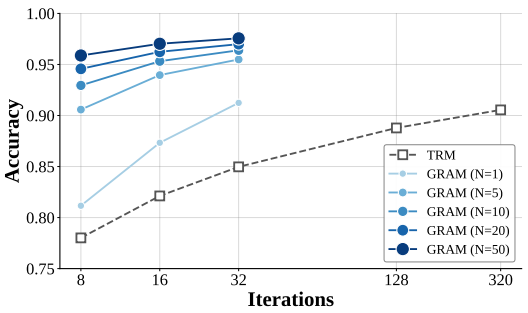


Figure 4: **Inference-time scaling on Sudoku-Extreme.** While both TRM and GRAM benefit from longer recursion (x-axis), GRAM additionally scales with parallel sampling (N = number of samples). Here, an iteration corresponds to a supervision step ($= T$ latent transitions, $z_t \rightarrow z_{t+T}$)

3.2 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 (Vaswani et al., 2017)), recursive models (HRM (Wang et al., 2025), TRM (Jolicoeur-Martineau, 2025)), and generative models (Autoregressive Transformer (AR), MDLM (Sahoo et al., 2024)). 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 E.1.

Table 1: **Evaluation on N-Queens and Graph Coloring benchmarks.** Rec. and Gen. indicate whether the model uses recursive computation and generative sampling, respectively. GRAM uniquely combines both properties, inheriting the reasoning strength of recursive models while leveraging generative sampling for solution diversity. 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	39.1	12.4	13.0	1.4	183	19.7	204	6.7
Direct Pred (32 layers)	✗	✗	100M	39.5	12.5	12.9	1.3	154	19.7	265	4.8
HRM	✓	✗	27M	80.7	25.5	54.0	5.6	111	21.4	162	7.2
TRM	✓	✗	7M	72.9	23.1	24.3	2.5	112	21.9	175	6.5
AR	✗	✓	10.6M	96.5	85.2	88.0	52.3	32	82.5	64	40.1
MDLM	✗	✓	12.6M	97.3	87.8	77.7	48.1	3	**87.1**	19	46.6
GRAM (Ours)	✓	✓	**10M**	**99.7**	**88.1**	**90.3**	**53.7**	**1**	86.0	**3**	**53.5**

Deterministic Recursion Fails on Multi-Solution Tasks. Table 1 reveals that deterministic recursive models (HRM, TRM) structurally cannot capture multiple solutions, with coverage below 25% across all tasks. Further illustrating this limitation, as the number of valid solutions increases, TRM and HRM exhibit sharp accuracy degradation, whereas GRAM maintains consistent performance regardless of solution count (see Appendix F.3). This confirms that deterministic latent updates cause mode collapse when multiple valid outputs exist for the same input. Additional coverage analysis is provided in Appendix F.4.

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 96.5% (AR) and 97.3% (MDLM). The gap is more pronounced on Graph Coloring, where GRAM reduces conflict edges to 1 and 3 on 8- and 10-vertex tasks, compared to 32 and 64 for AR. This demonstrates that recursive refinement enables stricter constraint satisfaction than generative sampling alone.

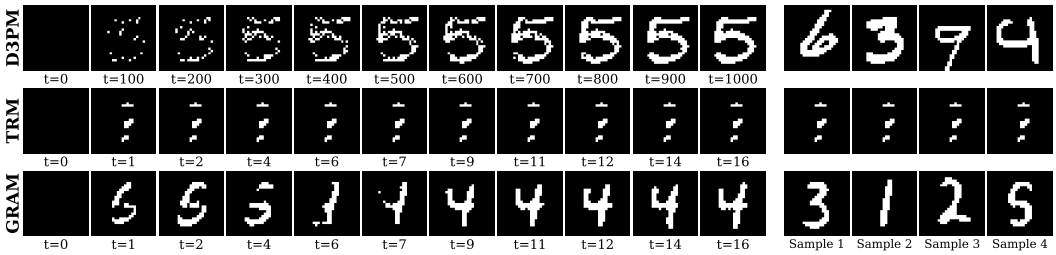
3.3 Exploring GRAM as Image Generator

Setup. To investigate GRAM's potential in diverse domains beyond reasoning tasks, we conduct experiments on binarized MNIST (Lecun et al., 1998), where pixel values are thresholded to 0 or 1. We formulate the generation task by defining the input x as a fully black image (all zeros) and the target y as the original MNIST image. We select D3PM (Austin et al., 2021) as a baseline, as it is a discrete diffusion model that similarly performs discrete pixel prediction. For evaluation, we measure Inception Score (IS) (Barratt & Sharma, 2018) and Fréchet Inception Distance (FID) (Heusel et al., 2017). To ensure a fair comparison with existing literature, FID is calculated using real samples from the original standard MNIST (0-255 pixel values).

Table 2: **Unconditional generation results on binarized MNIST.** We report IS (↑) and FID (↓). A step corresponds to a supervision step (T latent transitions, $z_t \to z_{t+T}$) 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 (↓)
D3PM-Uniform (1000 steps)	**1.91**	**69.78**
D3PM-Absorb (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**

Effective Modeling of Complex Data Distributions. Table 2 summarizes the quantitative results. While the deterministic baseline TRM suffers from complete mode collapse, producing invalid or repetitive outputs, GRAM generates high-quality samples comparable to specialized generative models like D3PM. This confirms that GRAM's recursive latent transitions can effectively model complex data distributions beyond symbolic reasoning tasks.



(a) Generation Process (b) Samples

Figure 5: **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.

Generation Quality Improves with More Recursion. As shown in Table 2, GRAM exhibits effective inference-time scaling. Although trained with only 16 steps, increasing the number of steps at inference time consistently improves image quality. This result indicates that GRAM successfully retains the iterative refinement advantage of recursive models within a generative framework. Figure 5 visualizes this generation process, showing how the model progressively refines the latent state to correct initial errors and produce cleaner target images. Additional samples are provided in Appendix F.6.

4 Related Works

Latent Reasoning. Latent reasoning addresses the inefficiencies of explicit Chain-of-Thought (CoT) by refining reasoning in a continuous space (Wei et al., 2022; Yao et al., 2023; Besta et al., 2024; Hao et al., 2024; Zhu et al., 2025; Gozeten et al., 2025). Several approaches leverage continuous state updates or internal recursive loops to scale test-time compute (Zhuang et al., 2025; Zhang et al., 2025; Butt et al., 2025; Hao et al., 2024; Shen et al., 2025; Yue et al., 2025; Geiping et al., 2025; Bae et al., 2025; Yang et al., 2023; Mohtashami et al., 2023; Bae et al., 2024). Recently, recursive reasoning models like HRM and TRM (Wang et al., 2025; Jolicoeur-Martineau, 2025) maintain predictions in latent space and iteratively refine them, yet they remain limited by their deterministic nature, which restricts exploration to a single fixed path. GRAM extends this paradigm by introducing stochastic transitions, enabling the explicit exploration of diverse reasoning trajectories.

Recursive Architectures. Recursive architectures, characterized by iterative state updates, have evolved from RNNs to weight-sharing Transformers capable of adaptive computation (Elman, 1990; Hochreiter & Schmidhuber, 1997; Cho et al., 2014; Dehghani et al., 2018; Yang et al., 2023; Lan et al., 2019; Elbayad et al., 2019; Graves, 2016; Mohtashami et al., 2023). Recent works utilize this property to scale inference-time compute, showing that deep recursive chains can outperform larger static models (Geiping et al., 2025; Bae et al., 2025; Wang et al., 2025; Jolicoeur-Martineau, 2025). GRAM leverages this foundation but distinguishes itself by formulating recurrence as a probabilistic process, supporting multi-path planning and generative sampling.

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. Only with 10M parameters, GRAM achieves competitive performance with much larger language models on ARC-AGI. Beyond solution-seeking, GRAM also demonstrates potential as an unsupervised learning approach through a recursion-based generative model over inputs, with generation quality improving over recursive refinement. 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). The authors at Mila acknowledge funding from the National Research Council (NRC), Samsung, and CIFAR. The research was enabled in part by computational resources provided by the Digital Research Alliance of Canada (`https://alliancecan.ca`) and Mila (`https://mila.quebec`). Minsu Kim acknowledges funding from the KAIST Jang Young Sil Fellow Program.

ETHIC STATEMENT

This paper presents GRAM, a generative latent-space recursive reasoning model that aims to advance the field of machine learning by enabling efficient, iterative reasoning and test-time scaling without relying on long token-level chain-of-thought sequences. The primary intended positive impact is to make strong reasoning and planning capabilities more accessible with substantially smaller models, which may reduce the barriers of compute and energy for research and deployment.

Potential negative impacts are expected to be similar to those of other general-purpose machine learning methods that improve reasoning capability and efficiency. As with many ML techniques, misuse is possible if applied in inappropriate contexts, and performance on controlled benchmarks may not fully reflect behavior under distribution shift or adversarial inputs. We encourage future work and any deployment to include careful evaluation across diverse settings, clear reporting of limitations, and standard safeguards appropriate to the application domain.

REFERENCES

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

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.

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.

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.

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

Yoshua Bengio et al. From system 1 deep learning to system 2 deep learning. In *Neural Information Processing Systems*, 2019.

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

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, pp. 17682–17690, 2024.

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

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

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.

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

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.

John P Coetzee, Micah A Johnson, Youngzie Lee, Allan D Wu, Marco Iacoboni, and Martin M Monti. Dissociating language and thought in human reasoning. *Brain Sciences*, 13(1):67, 2022.

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

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

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.

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

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.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.

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.

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.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

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

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

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.

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.

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

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

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

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

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.

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.

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.

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)*, pp. 879–885. IEEE, 2019.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

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

Kyle Mahowald, Anna A Ivanova, Idan A Blank, Nancy Kanwisher, Joshua B Tenenbaum, and Evelina Fedorenko. Dissociating language and thought in large language models. *Trends in cognitive sciences*, 28(6):517–540, 2024.

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.

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

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

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

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.

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

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.

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

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

Haoxiang Sun, Yingqian Min, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, Zheng Liu, Zhongyuan Wang, and Ji-Rong Wen. Challenging the boundaries of reasoning: An olympiad-level math benchmark for large language models. *arXiv preprint arXiv:2503.21380*, 2025.

Gladys Tyen, Hassan Mansoor, Victor Cărbune, Yuanzhu Peter Chen, and Tony Mak. Llms cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13894–13908, 2024.

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.

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.

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.

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

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

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.

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.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

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.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

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.

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

A USE OF LARGE LANGUAGE MODELS

During the preparation of this manuscript, the authors used large language models (e.g., Claude, GPT-5) to assist with refining prose, improving grammatical clarity, and enhancing readability. These tools were not used for generating scientific ideas, experimental design, data analysis, or drawing conclusions. All content was critically reviewed, verified, and revised by the authors, who take full responsibility for the final manuscript.

B ADDITIONAL METHOD DETAILS

B.1 INFERENCE-TIME SCALING OF GRAM

The generative latent transitions in GRAM enable the model to sample diverse latent reasoning trajectories $\{\tau^{(n)} = (z_0^{(n)} \to \ldots \to z_T^{(n)})\}_{n=1}^N$ conditioned on an input x. At inference time, this stochasticity allows efficient *parallel sampling* of N trajectories from the learned prior $p_\theta(\tau|x)$, each yielding a candidate solution $\hat{y}^{(n)} = f_{\text{dec}}(h_T^{(n)})$.

This parallel sampling capability is particularly beneficial when individual trajectories become trapped in suboptimal reasoning paths. While extending a single deterministic trajectory with additional reasoning steps often fails to escape poor local optima, sampling multiple stochastic trajectories in parallel substantially increases the probability of discovering higher-quality solutions.

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

$$\mathcal{L}_{\text{LPRM}} = \sum_{t=1}^{T} (v_\psi(h_t) - r)^2, \tag{14}$$

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

Selection Strategies. Given N candidate outputs $\{\hat{y}^{(n)}\}_{n=1}^N$, we consider two strategies for selecting the final prediction:

(i) Majority Voting: Select the most frequently predicted candidate:

$$\hat{y} = \arg\max_y \sum_{n=1}^{N} \mathbf{1}[\hat{y}^{(n)} = y]. \tag{15}$$

(ii) LPRM-Guided Selection: Choose the trajectory with the highest predicted value:

$$\hat{y} = \hat{y}^{(n^*)}, \quad n^* = \arg\max_n v_\psi(h_T^{(n)}). \tag{16}$$

Overall, this inference-time scaling scheme allows GRAM to efficiently explore multiple plausible reasoning paths in parallel, improving robustness and solution quality without incurring additional sequential computation cost.

C ADDITIONAL THEORETICAL SUPPORT

C.1 DERIVATION OF TRUNCATED ELBO

We derive the truncated ELBO used in Section 2.2 by analyzing which terms contribute to the gradient under truncation.

Full ELBO. From Section 2.2, the ELBO decomposes as:

$$\mathcal{L}_{\text{ELBO}}(x, y; \theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\log p_\theta(y|h_T, x)]}_{\text{likelihood}} - \underbrace{\sum_{t=1}^{T} \mathbb{E}_{q_\phi(\epsilon_{<t}|x,y)}[\text{KL}_t]}_{\text{KL terms}}, \tag{17}$$

where $\mathrm{KL}_t = \mathrm{KL}(q_\phi(\epsilon_t|u_t, y)\|p_\theta(\epsilon_t|u_t))$.

Dependency Analysis. At each step t, the low-level state is refined over K inner steps:

$$l_{t,k} = f_{\mathrm{L}}(h_{t-1}, l_{t,k-1}, e_x; \theta), \quad k = 1, \ldots, K, \tag{18}$$

with $l_{t,0} := l_{t-1}$ and $l_t := l_{t,K}$. The high-level state is then updated via:

$$u_t = f_{\mathrm{H}}(h_{t-1}, l_t; \theta), \quad \epsilon_t \sim q_\phi(\epsilon_t|u_t, y), \quad h_t = u_t + \epsilon_t. \tag{19}$$

Unrolling the recursion reveals that u_t depends on (i) θ , through f_{L} and f_{H} at all steps $1, \ldots, t$, and on (ii) ϕ , through ancestral samples $\epsilon_{<t} \sim q_\phi$. We write this dependency as $u_t = u_t(\epsilon_{<t}; \theta)$. Computing exact gradients requires backpropagation through all T recursive steps.

Applying Truncation. Following Jolicoeur-Martineau (2025), we detach the latent state at step $T-1$:

$$\tilde{h}_{T-1} = \mathrm{stopgrad}(h_{T-1}), \quad \tilde{l}_{T-1} = \mathrm{stopgrad}(l_{T-1}). \tag{20}$$

The final step ($t = T$) retains full gradient flow:

$$l_{T,k} = f_{\mathrm{L}}(\tilde{h}_{T-1}, l_{T,k-1}, e_x; \theta), \quad k = 1, \ldots, K, \tag{21}$$

$$u_T = f_{\mathrm{H}}(\tilde{h}_{T-1}, l_T; \theta), \quad h_T = u_T + \epsilon_T. \tag{22}$$

This truncation has the following effects:

(1) Likelihood term: Applying the chain rule for θ:

$$\nabla_\theta \mathbb{E}_q[\log p_\theta(y|h_T, x)] = \mathbb{E}_q\left[\frac{\partial \log p_\theta}{\partial h_T} \cdot \frac{\partial h_T}{\partial \theta}\right]. \tag{23}$$

Since $h_T = u_T + \epsilon_T$ and $u_T = f_{\mathrm{H}}(\tilde{h}_{T-1}, l_T; \theta)$ where l_T depends on θ through f_{L}, the gradient flows through the final step:

$$\frac{\partial h_T}{\partial \theta} = \frac{\partial u_T}{\partial \theta} \neq 0. \tag{24}$$

For ϕ, using the reparameterization $\epsilon_T = \mu_\phi(u_T, y) + \sigma_\phi(u_T, y) \odot \xi$ where $\xi \sim \mathcal{N}(0, I)$:

$$\nabla_\phi \mathbb{E}_q[\log p_\theta(y|h_T, x)] = \mathbb{E}_\xi\left[\frac{\partial \log p_\theta}{\partial h_T} \cdot \frac{\partial \epsilon_T}{\partial \phi}\right] \neq 0. \tag{25}$$

(2) KL terms for $t < T$: For each KL_t, the parameters appear directly in the distributions and indirectly through u_t:

$$\nabla_\theta \mathrm{KL}_t = \underbrace{\frac{\partial \mathrm{KL}_t}{\partial \theta}\bigg|_{u_t}}_{\text{direct}} + \underbrace{\frac{\partial \mathrm{KL}_t}{\partial u_t} \cdot \cancelto{0}{\frac{\partial u_t}{\partial \theta}}}_{\text{indirect (blocked)}} = 0, \tag{26}$$

$$\nabla_\phi \mathrm{KL}_t = \underbrace{\frac{\partial \mathrm{KL}_t}{\partial \phi}\bigg|_{u_t}}_{\text{direct}} + \underbrace{\frac{\partial \mathrm{KL}_t}{\partial u_t} \cdot \cancelto{0}{\frac{\partial u_t}{\partial \phi}}}_{\text{indirect (blocked)}} = 0. \tag{27}$$

The direct terms vanish because KL_t depends on θ and ϕ only through u_t (which conditions the distributions). The indirect terms vanish because u_t depends on $\tilde{z}_{T-1}$, which is detached.

(3) KL term for $t = T$: Unlike earlier steps, the distributions $p_\theta(\epsilon_T|u_T)$ and $q_\phi(\epsilon_T|u_T, y)$ have direct parameterization through their mean and variance networks:

$$\nabla_\theta \mathrm{KL}_T = \frac{\partial \mathrm{KL}_T}{\partial \theta}\bigg|_{u_T} + \frac{\partial \mathrm{KL}_T}{\partial u_T} \cdot \frac{\partial u_T}{\partial \theta} \neq 0, \tag{28}$$

$$\nabla_\phi \mathrm{KL}_T = \frac{\partial \mathrm{KL}_T}{\partial \phi}\bigg|_{u_T} \neq 0. \tag{29}$$

The direct term for θ is non-zero because $p_\theta(\epsilon_T|u_T) = \mathcal{N}(\mu_\theta(u_T), \sigma_\theta^2(u_T)I)$ is parameterized by θ. The indirect term is also non-zero because u_T retains gradient flow through f_{L} and f_{H}.

Truncated ELBO. Retaining only terms with non-zero gradients, we obtain:

$$\tilde{\mathcal{L}}_{\mathrm{ELBO}}(x, y; \theta, \phi) = \mathbb{E}_{q_\phi}[\log p_\theta(y|h_T, x)] - \mathrm{KL}(q_\phi(\epsilon_T|u_T, y)\|p_\theta(\epsilon_T|u_T)), \tag{30}$$

where $h_T = u_T + \epsilon_T$ and $u_T = f_{\mathrm{H}}(\tilde{h}_{T-1}, l_T; \theta)$ with $\tilde{z}_{T-1} = \mathrm{stopgrad}(z_{T-1})$. This objective trains the encoder, decoder, and the final recursion step (f_{L}, f_{H}, and the prior/posterior networks) while treating earlier steps as fixed context.

D Training and Architecture Details

D.1 Architecture Details.

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

Table 3: 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. Input tokens are mapped to embeddings via a token embedding layer, optionally concatenated with puzzle embeddings (for ARC (Chollet, 2019; Chollet et al., 2025)), and combined with positional encodings (RoPE) (Su et al., 2024). The embeddings are scaled by $\sqrt{D}$ and prepended with 16 puzzle embedding tokens (Wang et al., 2025).

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 (Shazeer, 2020) 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 (Jolicoeur-Martineau, 2025). 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.

Encoder and Decoder for Image Patches. In the MNIST (Lecun et al., 1998) 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 (Ryu, 2024; Peebles & Xie, 2023).

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 (LeCun et al., 2002; Krizhevsky

et al., 2012) with a 5×5 kernel and padding 2, a SiLU non-linearity (Elfwing et al., 2018), and Group Normalization (GN) (Wu & He, 2018). 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 4.

Table 4: 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 ($p = 2$) SiLU $\rightarrow$ GN(32)	$(B, D/2, H, W)$
	Conv2d 5×5 ($p = 2$) SiLU $\rightarrow$ GN(32)	$(B, D/2, H, W)$
3. Patch	Flatten Patches	$(B, N_p, P^2 \cdot \frac{D}{2})$
	Linear Projection	(B, N_p, D)

Hyperparameters. Following Wang et al. (2025); Jolicoeur-Martineau (2025), 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 (Vaswani et al., 2017) 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 $z_t \rightarrow z_{t+1}$, we use low-level (inner) steps $K = 6$ for Sudoku (Wang et al., 2025) and $K = 4$ for all other tasks, with high-level (outer) steps $T = 3$.

D.2 Training Details

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

- For *Sudoku* (Wang et al., 2025), 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* (Chollet, 2019; Chollet et al., 2025), 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 upper triangle of the adjacency matrix as $n(n+1)/2$ tokens, using 0=diagonal, 1=no-edge, 2=edge for inputs and $3 +$ color_id for output colors.
- For image generation on *MNIST* (Lecun et al., 1998), images are quantized and processed via CNN-based patchification (LeCun et al., 2002; Peebles & Xie, 2023), 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).

D.3 Training Details

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

- For *Sudoku* (Wang et al., 2025), the 9×9 grid is flattened row-by-row into 81 tokens with vocabulary size 11 (0=pad, 1=blank, 2–10=digits).

Table 5: 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	$n(n+1)/2$	6+	✗	Adjacency upper triangle
MNIST	169	3	✗	14×14 patches

- For *ARC-AGI* (Chollet, 2019; Chollet et al., 2025), 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 upper triangle of the adjacency matrix as $n(n+1)/2$ tokens, using 0=diagonal, 1=no-edge, 2=edge for inputs and $3 +$ color_id for output colors.
- For image generation on *MNIST* (Lecun et al., 1998), images are quantized and processed via CNN-based patchification (LeCun et al., 2002; Peebles & Xie, 2023), 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	$n(n+1)/2$	6+	✗	Adjacency upper triangle
MNIST	169	3	✗	14×14 patches

Training Details. We train all models using AdamW (Loshchilov & Hutter, 2017) 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. (2018) and Song & Ermon (2020). To prevent posterior collapse, we use a KL balance (Hafner et al., 2020; 2023) coefficient of 0.8. The number of deep supervision steps is $N_{\text{sup}} = 16$ for all tasks. 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

On Sensitivity of KL Coefficient. The KL coefficient β is a sensitive hyperparameter that requires task-specific tuning. During training, the posterior receives the ground-truth label y as input, creating a trade-off: if β is too high, the posterior collapses to the prior and fails to leverage label information; if β is too low, insufficient regularization allows label information to leak directly into the latent trajectory, causing overfitting. We find that the optimal β varies across datasets (e.g., 0.8 for Sudoku, 0.1 for others), and recommend tuning this hyperparameter for new tasks.

Table 8: KL coefficient β per task.

	Sudoku	ARC-AGI	N-Queens	Graph Coloring	MNIST
β	0.1	0.04/0.1	0.07/0.045	0.5/0.45	0.07

E Additional Details of Experiment Setup

E.1 Multi-solution Puzzle Tasks

E.1.1 N-Queens Problem

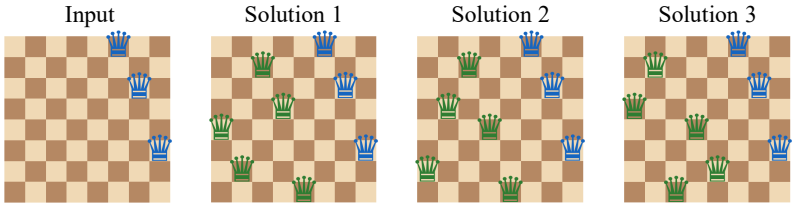


Figure 6: **Example of an 8x8 N-Queens puzzle instance.** In this example, 5 queens are removed from the full board, leaving 3 queens. The model must find the positions of the remaining queens. This configuration admits exactly 3 valid solutions.

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 6 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 7.

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).

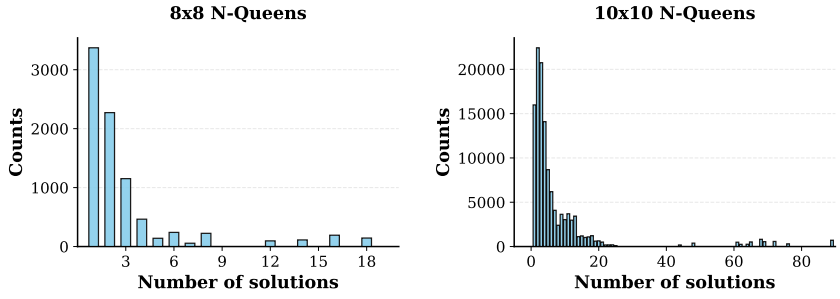


Figure 7: **Distribution of the number of valid solutions for generated N-Queens instances.** The dataset covers a wide range of solution counts, testing the model's ability to recover multiple valid outputs.

E.1.2 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 8 illustrates an example instance with $N = 8$ nodes and $k = 3$ colors.

Graphs are generated using the Erdős–Rényi random graph model (Erdős & Rényi, 1964), following the generation pipeline from GNN-GCP (Lemos et al., 2019). 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 9.

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.

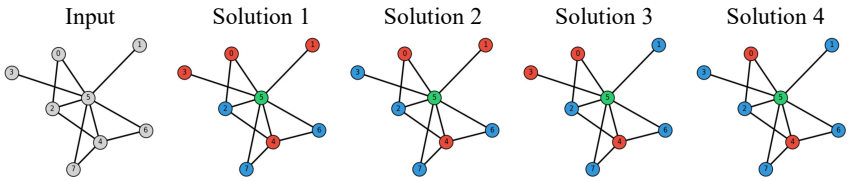


Figure 8: **Graph Coloring Example**

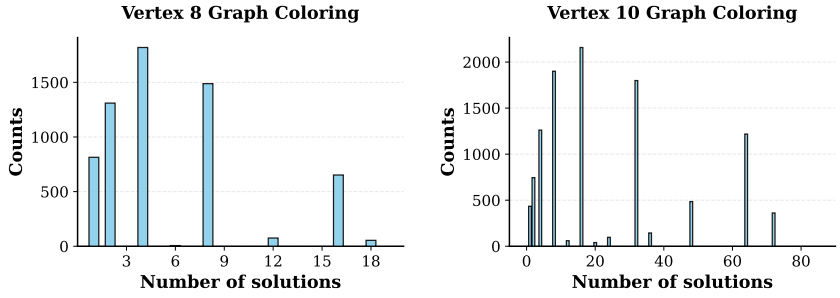


Figure 9: **Distribution of the number of valid solutions for generated N-Queens instances.** The dataset covers a wide range of solution counts, testing the model's ability to recover multiple valid outputs.

F ADDITIONAL EXPERIMENT RESULTS

F.1 ADDITIONAL RESULTS ON CHALLENGING PUZZLE BENCHMARKS

Table 9 reports test accuracy on three challenging puzzle benchmarks. Here we provide additional observations complementing the main text.

Comparison Among Recursive Models. GRAM consistently outperforms all prior recursive models across benchmarks. On Sudoku-Extreme, GRAM (97.0%) improves over TRM (87.4%) by

Table 9: **Test accuracy (%) on Challenging Puzzle Benchmarks.** GRAM significantly outperforms prior recursive models and matches GPT-5 (low) level performance with only 10M parameters. All recursive model scores were obtained at 16 iterations (48 transitions).

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
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

+9.6% and over HRM (55.0%) by +42.0%. On ARC-AGI-1 and ARC-AGI-2, the improvements are +7.4% and +3.3% over TRM, respectively. Notably, GRAM achieves these gains with 10M parameters, compared to 27M for HRM. The Direct Prediction baseline (0% on Sudoku, 0% on ARC-2) confirms that recursion is essential for these tasks.

Comparison with Large Reasoning Models. All tested LRMs, including o3-mini-high and Deepseek-R1 (671B), score 0% on Sudoku-Extreme, while GRAM achieves 97.0%. On ARC-AGI-2, GRAM (11.1%) surpasses GPT-5.2 (low) (9.7%) despite using orders of magnitude fewer parameters. On ARC-AGI-1, GRAM (52.0%) approaches GPT-5.2 (low) (55.7%). Among LRMs, scaling parameters alone does not guarantee better performance—Deepseek-R1 (671B) scores lower than o3-mini-high on ARC-AGI-1 (15.8% vs. 34.5%). More recent models such as Gemini 3 Pro (75.0% on ARC-1, 31.1% on ARC-2) remain substantially ahead, suggesting room for further improvement in recursive approaches.

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

To investigate the effect of GRAM's sampling on the ARC-AGI-1 benchmark as well, 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 10, removing augmentations causes a performance decline for both GRAM and TRM compared to the values reported in Table 9. 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 3.2, 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 11 presents the results across varying augmentation levels (Aug=0 to Aug=50).

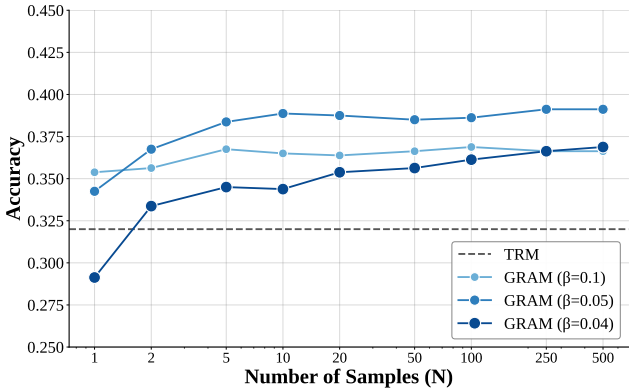


Figure 10: **Effect of sampling on ARC-AGI-1 without data augmentation.** To isolate the internal sampling effect, both models are evaluated on original problem instances without 1,000 augmentations. While removing augmentations causes an initial performance drop, GRAM exhibits robust scaling through generative sampling as the number of parallel samples N increases, outperforming the TRM baseline.

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.

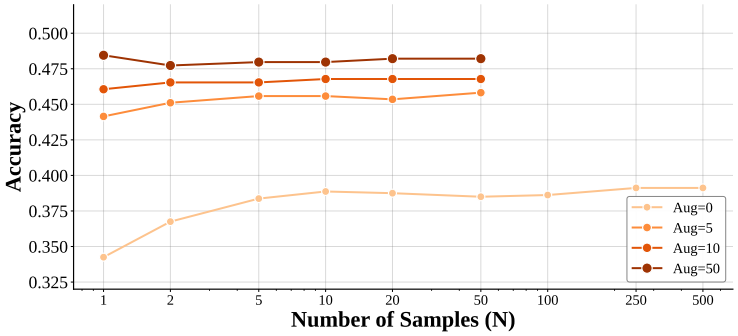


Figure 11: **Effect of augmentation on sampling efficiency.** With limited augmentation (Aug=0), parallel sampling provides consistent gains. As augmentation increases, sampling benefits diminish—at Aug= 50, performance saturates regardless of sample count, suggesting augmentation and sampling serve complementary roles in capturing solution diversity.

F.3 Impact of the Number of Solutions on Accuracy

To further investigate how the number of valid solutions affects model performance, we analyze the accuracy of models on the N-Queens (8×8) task across varying numbers of ground-truth solutions. In highly ambiguous problem settings where multiple valid configurations exist for a single input, deterministic recursive models face a significant structural disadvantage.

As illustrated in Figure 12, conventional deterministic models, such as TRM and HRM, exhibit a severe performance degradation as the number of possible solutions increases. Because these models are constrained to a single deterministic reasoning trajectory, they struggle to resolve the ambiguity of multiple valid targets. This structural limitation forces the model to average over distinct solution paths, often resulting in conflicting latent updates and incorrect final predictions.

In contrast, GRAM explicitly models a distribution over plausible reasoning paths. By leveraging stochastic latent transitions, GRAM effectively navigates the multi-solution landscape without suffering from mode collapse. Consequently, GRAM maintains consistent, near-perfect accuracy regardless of the number of valid solutions, highlighting the robustness of generative recursion in highly ambiguous reasoning tasks.

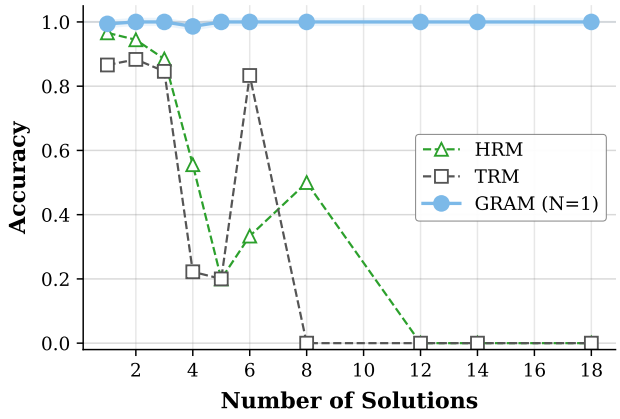


Figure 12: **Accuracy across number of solutions in N-Queens (8×8).** Conventional deterministic recursive models suffer a sharp performance drop as the number of possible solutions increases, whereas GRAM maintains consistent performance.

F.4 Solution Coverage Analysis

We analyze the ability of GRAM to capture the diversity of the solution space compared to deterministic baselines. Figure 13 presents the solution coverage on 8×8 and 10×10 N-Queens tasks with respect to the total number of valid ground-truth solutions.

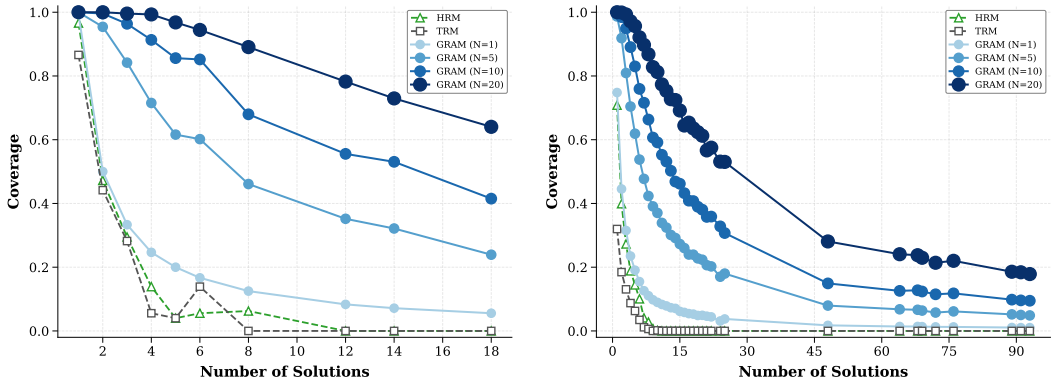
As shown in Figure 13, deterministic recursive models (HRM and TRM) exhibit a sharp decline in coverage as the number of possible solutions increases. Since these models are constrained to a single fixed reasoning trajectory, they structurally fail to explore alternative paths, resulting in severe mode 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.

F.5 Ablation Study

We ablate key design choices of GRAM on Sudoku-Extreme and N-Queens (8×8) using 5 samples. Table 10 summarizes the results.

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 (9). 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.



(a) N-Queens 8×8 (b) N-Queens 10×10

Figure 13: **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.

Table 10: **Ablation study on Sudoku-Extreme and N-Queens (8×8).** We evaluate with 5 samples. Both stochasticity and learned guidance are essential—removing either significantly degrades performance. Naive approaches to adding stochasticity to TRM fail to match GRAM.

Model Variant	**Sudoku**	**N-Queens**
GRAM (Ours)	**93.96**	**99.69**
w/o stochastic guidance (= TRM)	82.87	72.91
w/o guide (stochasticity only)	94.88	50.27
w/o stochasticity (guide only)	0.00	0.00
w/ direct prediction	63.43	61.44
TRM w/ stochastic decoding	82.87	71.66
TRM w/ random initialization	78.53	71.82

Residual Guidance Outperforms Direct Prediction. We compare our residual formulation $h_t = u_t + \epsilon_t$ in Equation (7) against directly sampling $h_t \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2 I)$. Direct prediction degrades performance (63.43% on Sudoku), indicating that preserving the deterministic update u_t with learned perturbations is crucial for stable exploration.

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 (2) *random initialization*, which samples z_0 from a Gaussian $\mathcal{N}(0, I)$ at each inference. Neither improves performance (82.87% and 78.53% on Sudoku), demonstrating that GRAM's gains stem from the variational framework rather than mere randomness.

F.6 Additional Generated Image Samples

In this section, we provide further qualitative results demonstrating GRAM's capability in unconditional image generation. Figure 14 presents a diverse set of samples generated on the binarized MNIST dataset, visualized across the recursive inference steps $t = 0$ to $t = 16$.

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 $t = 16$.

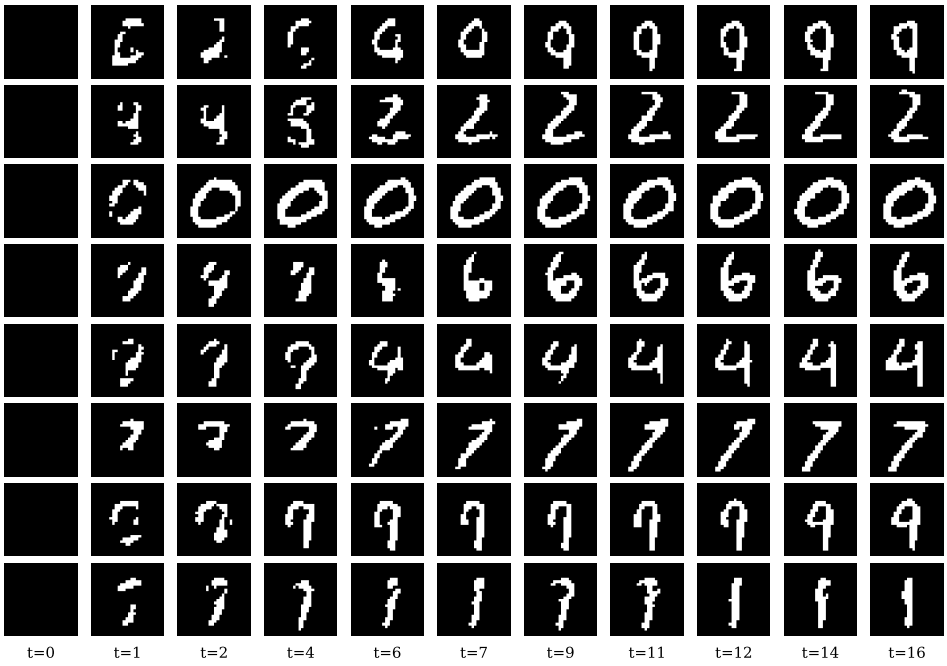


Figure 14: **Additional generated samples from GRAM.** We provide 8 additional samples generated unconditionally on binarized MNIST using GRAM. Each row represents a single generated sample, visualized across its recursive refinement process.

F.7 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 hh h at each supervision step throughout the recursion process. For visualization, we project these latent vectors into 2D using PCA (Pomerantsev, 2014) and interpolate unobserved states via K-D tree (Bentley, 1975) to construct a continuous loss landscape.

Figures 15 and 16 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.

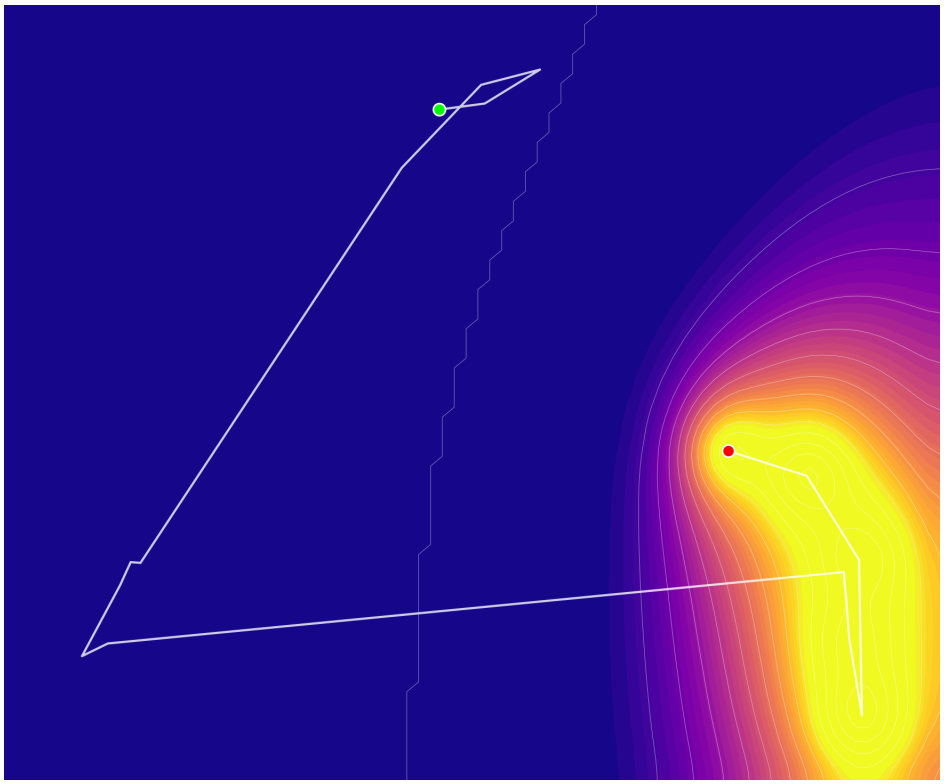


Figure 15: **Latent reasoning trajectory of TRM.** The red dot indicates the initial state h_0 and the green dot indicates the final state h_T. Background color represents the loss landscape: bright yellow corresponds to high loss regions, while dark blue indicates low loss (optimal) regions. TRM follows a single deterministic path with no ability to escape suboptimal trajectories.

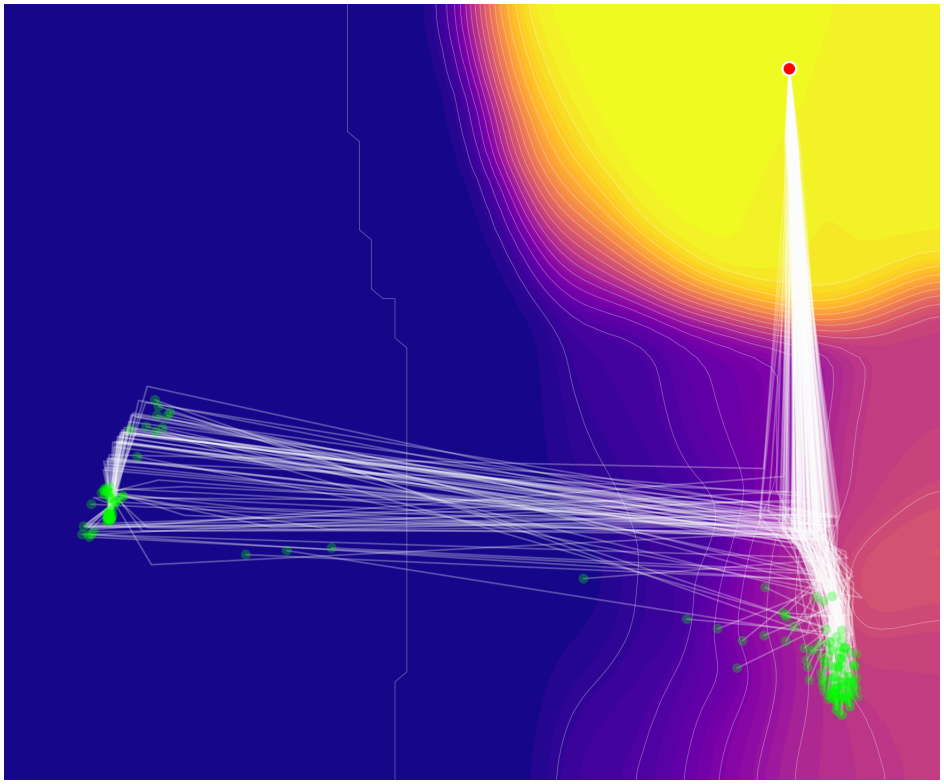


Figure 16: **Latent reasoning trajectories of GRAM (50 samples).** Using the same visualization scheme as Figure 15, we show 50 sampled trajectories from GRAM. The stochastic guidance enables diverse exploration of the latent space: while some trajectories converge to local minima (right bottom), others successfully reach the global optimum (left middle), demonstrating how parallel sampling improves solution discovery.