LEARNING RECURSIVE REASONERS WITHOUT SOLUTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive reasoning models obtain effective depth by repeatedly applying small networks, but are typically trained on ground-truth solutions. We show that such models can instead be learned without solution supervision, using only a black-box verifier or energy function that scores candidate solutions. Our solution-free variant combines latent deterministic recurrence with stochastic refinement of explicit candidate assignments and is trained from terminal verifier feedback alone. On constraint satisfaction and combinatorial optimization problems, such as *Graph Coloring*, *Maximum Independent Set* and *Sudoku*, the resulting models are effective solvers, matching the state of the art. To our knowledge, we provide the first solution-free model to exceed 90% on *Sudoku-Extreme*. We further find that latent recurrence is a test-time scaling axis complementary to explicit refinement and parallel search, and that the compute-optimal allocation across these axes is problem-dependent. Finally, our results show that more training-time recurrence does not necessarily improve depth extrapolation, whereas truncating credit assignment to the final recursive cycle yields operators that keep improving well beyond their training depth.

1 INTRODUCTION

Many neural approaches to structured reasoning obtain their problem-solving ability from *iterative computation*. Rather than mapping an instance to a solution in a single feed-forward pass, they repeatedly refine an internal representation or a candidate answer. Recursive reasoning models (RRMs), such as the Hierarchical Reasoning Model (HRM) and Tiny Recursive Model (TRM), make this principle explicit by repeatedly applying small, weight-shared neural networks, obtaining large effective computational depth with comparatively few parameters (Wang et al., 2025; Jolicoeur-Martineau, 2025). Recent extensions improve symmetry handling, stochastic exploration, and generative reasoning (Freinschlag et al., 2026; Sghaier et al., 2026; Baek et al., 2026). Despite these advances, current RRMs learn their reasoning dynamics from known target solutions. While they do not require explicit reasoning traces, training assumes access to correct input–solution pairs and repeatedly supervises predictions against these targets.

For hard combinatorial problems, however, producing such targets may require solving the very instances used for training, which implies a requirement of the availability of scalable solvers to generate sufficient amounts of training data. In addition to this problem, many combinatorial problems admit large sets of valid solutions, which makes supervision with one arbitrarily selected target unnecessarily restrictive. Solution-free neural solvers avoid both issues by learning from the problem objective alone. Earlier solution-free approaches train graph neural networks by optimizing differentiable surrogates of the combinatorial objective and constraints (Karalias & Loukas, 2020), including recurrent message passing over soft assignments (Tönshoff et al., 2021), or with policy gradients over search trajectories (Tönshoff et al., 2023). ConsFormer trains a Transformer to improve candidate assignments using differentiable surrogates of constraints and iterates it at test time (Xu et al., 2025). Diffusion-based samplers learn a stochastic sequence of refinement decisions from an energy function alone and optimize it with policy gradients (Sanokowski et al., 2024; 2025). In these methods, iterative computation is tied to revisions of the candidate assignment, or to a recurrent state that is updated once per revision. To our knowledge, none of them separates the amount of weight-shared latent computation within a revision from the number of revisions.

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

This raises the central question of this work: how should a solution-free solver split its iterative computation between weight-shared latent recursion within a revision and the number of revisions, and can the first part be learned at all when the only training signal is the energy of the final candidate? We study this setting under a particularly weak form of supervision: for a problem instance I and candidate assignment x, training has access only to an efficiently evaluable, potentially non-differentiable energy $E_I(x)$ that measures solution quality or constraint violations. No ground-truth solutions, intermediate targets, or reasoning trajectories are provided. We refer to this setting as *unsupervised recursive reasoning*.

We instantiate this idea as uTRM, an unsupervised variant of TRM (Jolicoeur-Martineau, 2025) that acts as a stochastic recurrent refinement policy (Fig. 1). Given a discrete candidate solution, uTRM performs multiple hierarchical recursive updates in a continuous latent space before decoding into a distribution over an updated solution variable assignment from which the revised solution is sampled. Repeated application yields a trajectory of candidate solutions, while the computation within each refinement step remains latent and weight-shared. We train the resulting policy using only the energy of the final candidate and policy gradients. In contrast to diffusion-derived data-free objectives, our default objective does not prescribe a stochastic forward process or provide transition-level process rewards, i.e. the recursive reasoner must learn useful iterative computation from terminal verifier feedback alone.

We evaluate uTRM on Graph Coloring, Maximum Independent Set (MIS) and Sudoku. Although it never observes a target solution, uTRM matches or exceeds specialized solution-free solvers in most settings. It sets a new state of the art for generalization from 50 to 100 vertices in 5-color Graph Coloring (56.64% versus 55.67% for ConsFormer) and solves 94.17% of Sudoku-Extreme puzzles, to our knowledge the first result above 90% for a model trained without solutions. The setting is particularly natural for Graph Coloring and MIS, where many assignments are valid and the energy rewards solutions within the entire feasible set rather than a given specifically selected solution target.

An important property of recursive models is that computation can be increased at inference time simply by applying the shared reasoning operator for additional cycles. Recent work has independently demonstrated the value of recursive computation in supervised masked diffusion models, where recursive denoiser applications improve parameter efficiency and can partially substitute for additional denoising steps (Carballo-Castro et al., 2026). Our focus is different: we ask whether such a reusable reasoning operator can *emerge without target solutions*, and how its test-time scaling compares with the explicit stochastic refinement used by solution-free iterative solvers.

We find that latent recursion provides a test-time scaling axis that differs from adding refinement steps. On Graph Coloring, increasing the recursion depth beyond its training value yields much larger gains than extending the refinement trajectory, and in test time scaling uTRM overtakes a depth- and architecture-matched Transformer with roughly 6 times as many parameters. On MIS, recursion alone is not enough. The aforementioned Transformer keeps its lead when only the recursion depth is increased in uTRM. However, uTRM closes the gap to the Transformer when recursion and refinement are scaled in an optimized way together. The compute-optimal resource allocation therefore uses both axes.

Obtaining an iterable recurrent operator from training is not unconditional on the training specifics. Recurrent networks trained end to end can degrade when iterated beyond their training depth, and in supervised settings this "overthinking" can be mitigated by progressive training from detached intermediate states (Bansal et al., 2022) or by promoting convergence to input-dependent fixed points (Anil et al., 2022; Huang et al., 2026). Similarly, we find that credit assignment is decisive in the unsupervised setting as well. Backpropagating only through the final recursion cycle, as in TRM, yields operators that keep improving up to about eight times beyond their training depth on Graph Coloring, whereas full backpropagation through the same recurrence specializes the model to its training horizon. Training a solver to perform well at a fixed depth and training a reusable iterative operator are therefore in practice distinct training objectives.

Our main contributions are:

- We introduce *unsupervised Tiny Recursive Model (uTRM)*, a reasoning model that is trained from a black-box terminal verifier without access to ground-truth solutions or intermediate

reasoning supervision. We demonstrate that uTRM performs strongly on Graph Coloring, Maximum Independent Set, and Sudoku, including several state-of-the-art results.

- We show that latent recurrence provides a complementary inference-time scaling axis to explicit stochastic refinement and parallel search. The characteristics of this axis is task-dependent: recursive depth scales particularly strongly on Graph Coloring, while MIS benefits more from additional discrete refinement.

- We characterize the conditions under which recursive test-time depth scaling emerges in the Reinforcement Learning setting of uTRM. Training recurrence and, in particular, truncated recursive credit assignment strongly affect whether the learned reasoning operator remains useful beyond its training horizon.

2 Related Work

Recursive reasoning models. Recursive reasoning models obtain computational depth by repeatedly applying weight-shared neural modules. HRM introduced a hierarchical architecture with interacting high- and low-level recurrent states (Wang et al., 2025), and TRM simplified it to a single small network while improving accuracy on structured reasoning benchmarks (Jolicoeur-Martineau, 2025). Subsequent work incorporates symbol equivariance (Freinschlag et al., 2026), stochastic test-time exploration (Sghaier et al., 2026) and probabilistic latent trajectories (Baek et al., 2026). All of these models learn their reasoning dynamics from target solutions. We study whether a recursive reasoning operator can be learned when training provides only the energy of a final candidate.

Unsupervised neural combinatorial solvers. A separate line of work learns combinatorial solvers from the problem objective rather than from precomputed solutions. Early approaches train graph neural networks by recurrent message passing on differentiable constraint relaxations (Tönshoff et al., 2021) or with policy gradients to guide a search over assignments (Tönshoff et al., 2023). ConsFormer trains a Transformer to iteratively improve assignments using differentiable surrogates of CSP constraints (Xu et al., 2025) and has been extended with large neighborhood search (Xu et al., 2026a). Its training requires constraint structure from which a differentiable objective can be constructed. Energy-based samplers need only an unnormalized score of candidate solutions. LTFT trains conditional GFlowNets that construct a solution one decision at a time such that complete solutions are sampled in proportion to an exponentiated reward (Zhang et al., 2023). DiffUCO instead learns a discrete diffusion model that updates all variables at every step from the problem energy alone (Sanokowski et al., 2024), and SDDS scales this approach to long trajectories with policy gradients (Sanokowski et al., 2025). BloGDiT combines blocked Gibbs updates with Transformer-based diffusion samplers (Xu et al., 2026b), and Combinatorial Adjoint Matching (CAM) casts unsupervised discrete diffusion as stochastic optimal control over continuous-time Markov chains, replacing policy-gradient and importance-weighted estimators by trajectory-level adjoint signals (Feng et al., 2026). In all of these methods, sequential computation is spent on explicit decisions and each individual decision is made by a single forward pass of the network. We explore the impact of recursive computation for these decisions in the same unsupervised problem setting.

Test-time depth in recursive networks. Weight sharing allows a recursive network to be iterated more often at test time than during training. Recursive networks trained on easy instances can solve harder ones with additional iterations (Schwarzschild et al., 2021), but this extrapolation degrades when many iterations are applied unless the input is re-injected at every iteration and the gradient is decoupled from early iterations during training (Bansal et al., 2022). Equilibrium models extrapolate better when they converge to the same fixed point regardless of initialization (Anil et al., 2022), and Equilibrium Reasoners scale supervised recursive reasoners to far more iterations than seen in training by shaping such attractors (Huang et al., 2026). Recurrent-depth language models combine randomized training depth with truncated backpropagation (Geiping et al., 2025). Recursive Masked Diffusion Models apply recursion within each denoising step and show that it can partially substitute for additional denoising steps under matched compute (Carballo-Castro et al., 2026). All of these works learn from target solutions or data samples. We study depth extrapolation when the operator is learned from terminal energy feedback alone, and we compare recursion with explicit stochastic refinement and parallel sampling as competing uses of the same compute.

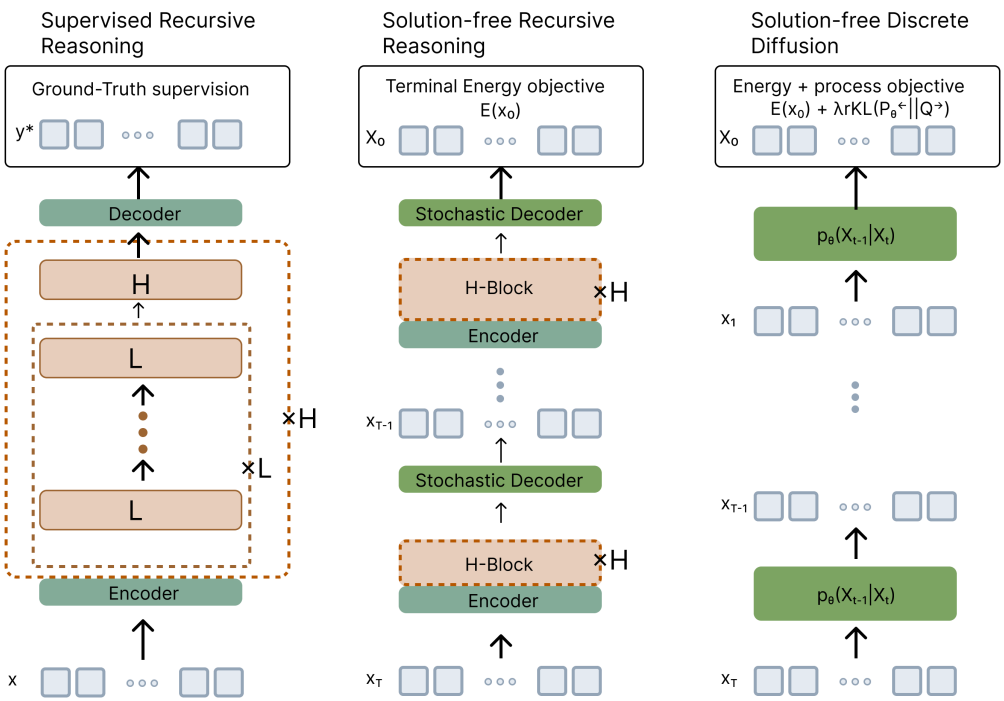

Figure 1: Comparison of supervised recursive reasoning (left), our solution-free recursive setting (middle), and solution-free discrete diffusion (right). Our method trains only from the terminal energy $E(x_0)$, without ground-truth solutions or diffusion-process supervision. Here $\times H$ and $\times L$ denote recurrent updates.

3 METHOD

For each problem instance I, we are given a discrete solution space $\mathcal{X}_I$ and an efficiently evaluable energy function

$$E_I : \mathcal{X}_I \to \mathbb{R}, \tag{1}$$

where lower energy corresponds to a better candidate. For the constraint satisfaction problems we consider, $E_I(x) = 0$ holds exactly only for feasible assignments. During training we observe problem instances and may evaluate E_I on arbitrary candidates, but we never observe a ground-truth solution $y^\star$. At test time E_I remains available and is used to select among parallel samples. We use the terms energy and verifier interchangeably, although for MIS the energy scores solution quality rather than only feasibility.

Our goal is to learn a recursive solver that maps an initially unstructured candidate to a low-energy solution. The model combines two forms of iterative computation: recursive refinement in continuous latent space and stochastic revision of the explicit discrete assignment. Importantly, only the latter is directly evaluated by the verifier.

3.1 RECURSIVE REASONING POLICY

In the following, we describe how uTRM generates solution candidates and how it is trained. Let I denote a problem instance and x_t the current discrete candidate assignment at refinement step t. The initial assignment x_T is sampled from a simple factorized prior. Task constraints enter the reasoning network through instance-dependent attention masks or biases derived from I (see App. B.3 for details).

Within each discrete refinement step, uTRM retains the latent recursive computation of TRM: a high-level solution representation z_H and a low-level latent reasoning state z_L are updated repeatedly using a shared reasoning network f_θ. At the beginning of each refinement step, the current candidate

x_t is embedded as $\mathbf{x}_t$, while the latent states z_H and z_L are reinitialized (see App. B.3 for details). The reasoning network, conditioned on the problem structure through its attention relations, first updates the low-level state L times,

$$z_L \leftarrow f_\theta(z_H + z_L + \mathbf{x}_t; I), \qquad \ell = 1, \ldots, L, \tag{2}$$

followed by one update of the high-level state,

$$z_H \leftarrow f_\theta(z_L + z_H; I). \tag{3}$$

Here, the dependence on I denotes structural conditioning through attention masks or biases rather than an explicit instance embedding. As in TRM, we refer to these $L + 1$ applications of the shared network as one high-level recursion cycle. Repeating this procedure H times yields a final high-level representation $z_H^{(H)}$.

At this point uTRM departs from supervised TRM (Jolicoeur-Martineau, 2025). In TRM, $z_H^{(H)}$ is decoded and compared with a known target solution $y^\star$. In our setting no such target is available. Instead, the decoded prediction defines a stochastic policy over the next discrete candidate assignment. A decoder f_{out} produces categorical logits

$$\ell_t = f_{\text{out}}\left(z_H^{(H)}\right), \tag{4}$$

from which the next assignment is sampled,

$$x_{t-1,i} \sim \text{Cat}\left(\text{softmax}(\ell_{t,i})\right), \qquad t = T, \ldots, 1. \tag{5}$$

A single policy evaluation therefore contains $H(L+1)$ applications of the shared reasoning network. If f_θ contains n_{layers} Transformer layers, one discrete refinement step has effective depth

$$H(L + 1)n_{\text{layers}}. \tag{6}$$

After sampling x_{t-1}, the latent states are reinitialized before the next refinement step. Consequently, the discrete assignment forms the only state propagated between successive policy evaluations,

$$x_T \to x_{T-1} \to \cdots \to x_0, \tag{7}$$

while the H recursive cycles within each step perform latent computation before producing the next candidate. Problem constraints remain available at every policy evaluation through the fixed, instance-dependent attention structure.

3.2 Solution-free training from terminal verifier feedback

We train the recursive policy using the energy of its final candidate. Starting from an initial assignment x_T sampled from a simple factorized prior, the policy produces a trajectory

$$\tau = (x_T, x_{T-1}, \ldots, x_0). \tag{8}$$

In our experiments the initial assignment is sampled uniformly from the domain of each decision variable.

The reward is zero for all intermediate transitions and given only after the final refinement step,

$$r_t = \begin{cases} 0, & t > 1, \\ -E_I(x_0), & t = 1. \end{cases} \tag{9}$$

The learning objective is therefore simply

$$J(\theta) = \mathbb{E}_{I,\tau\sim\pi_\theta}\left[-E_I(x_0)\right]. \tag{10}$$

Because E_I need not be differentiable, gradients can in general not be propagated directly through the verifier. Instead, we optimize the stochastic refinement policy with policy gradients and use PPO (Schulman et al., 2017) in all experiments (see App. B.2). The sampled discrete trajectory provides the actions to which the terminal return is attributed, i.e. no ground-truth solution, differentiable relaxation of the constraints, or supervision of the latent states is required. This objective deliberately makes no assumptions about how a solution should be reached. In particular, the model is not trained to reproduce a prescribed corruption or denoising process like in diffusion-based solvers.

3.3 TRUNCATED RECURSIVE CREDIT ASSIGNMENT

Backpropagating through all H recursion cycles makes activation memory grow linearly with recursion depth. Following HRM and TRM (Wang et al., 2025; Jolicoeur-Martineau, 2025), we retain gradients only through the final recursion cycle. The first $H-1$ cycles run without gradient tracking and initialize the final cycle. During each PPO update minibatch, the L- and H-cycles are re-executed under the current parameters, recomputing the latent states z_H and z_L. The backpropagation graph therefore contains $L+1$ applications of f_θ regardless of H. We find that uTRMs trained through all recursion cycles stop improving when evaluated with more cycles than used in training, whereas truncated training yields operators that keep improving beyond their training depth (Sec. 4.4).

3.4 INFERENCE-TIME COMPUTATION

The model exposes three independent ways to increase inference-time computation.

Recursive depth H. Increasing H applies the same latent reasoning operator additional times before producing the next discrete candidate. This increases deterministic computation without adding parameters or increasing the number of stochastic decisions.

Discrete refinement depth T. Increasing T gives the solver additional opportunities to revise its explicit candidate assignment. Unlike increasing H, every additional refinement step introduces a new categorical sampling operation.

Search breadth K. Finally, we may sample K independent trajectories and return the candidate with lowest energy. This increases total computation but does not increase the sequential depth of an individual trajectory.

For a model with n_{layers} layers in its shared reasoning core, a single trajectory requires

$$C_{\text{seq}} = TH(L+1)n_{\text{layers}} \tag{11}$$

Transformer-layer evaluations. Sampling K trajectories yields total computation

$$C_{\text{total}} = KC_{\text{seq}}. \tag{12}$$

We use these quantities to compare different allocations of inference compute, while additionally reporting parameter counts where applicable B.5.

4 EXPERIMENTS AND RESULTS

We evaluate three questions corresponding to the central claims of this work: (i) can a recursive reasoner be trained from a black-box verifier without target solutions across structurally different combinatorial problems; (ii) does latent recurrence provide an efficient and complementary axis for increasing test-time computation relative to explicit discrete refinement and parallel search; and (iii) which training choices determine whether the learned recursive operator remains useful when iterated beyond its training depth. We evaluate these questions on Graph Coloring, Maximum Independent Set (MIS), and Sudoku.

4.1 EXPERIMENTAL SETUP

All models are trained without access to ground-truth solutions. Training uses only problem instances and the corresponding energy functions defined in App. B.1. Our default recursive model uses $H_{\text{train}} = 2$ high-level cycles, $L = 2$ low-level updates per high-level cycle, and a shared Transformer core with $n_{\text{layers}} = 2$ layers. Unless stated otherwise, models are trained with terminal verifier feedback as described in Sec. 3.2.

For the controlled architecture comparison on Graph Coloring, we train a depth-matched 12-layer flat Transformer with the same terminal-verifier objective as uTRM. Its effective Transformer depth matches that of uTRM at the training configuration, but it uses untied parameters across layers rather than recursive weight sharing, and thus has 6 times as many trainable parameters. On MIS,

we instead compare against SDDS-rKL, a strong solution-free stochastic baseline in combinatorial optimization whose diffusion-derived reverse-KL objective augments the terminal reward with transition-level process rewards (see App. A.1). We use these MIS comparisons as a performance- and compute-matched reference. We separately study the effect of process rewards in our objective ablation (App. C.1).

Unless stated otherwise, evaluation returns the lowest-energy candidate among K independent trajectories. For direct comparison with ConsFormer (Xu et al., 2025) and BloGDiT (Xu et al., 2026b), we train and evaluate on their established datasets for Graph Coloring. Because these benchmark splits were also used for evaluation and checkpoint selection, we additionally test our GC models on an independently generated held-out split. These results are reported in Table 6 and App. C.4. The central GC50 architecture comparison and gradient-truncation experiment use three independent training seeds. Exploratory depth sweeps use a single training seed unless stated otherwise.

4.2 Can recursive reasoners be trained without solutions?

Table 1 shows that terminal verifier feedback allows learning effective recursive solvers across all three problem families.

On Graph Coloring, uTRM matches the strongest reported GC50 result, reaching 83.00% best-of-20 accuracy while using approximately six times fewer Transformer-layer evaluations than BloGDiT. The learned recursive operator also transfers to larger instances: on GC50→GC100, increasing recursive depth raises accuracy to 56.64%, improving the previously reported best solution-free results at substantially lower inference compute. On the in-distribution 10-color GC100 benchmark, uTRM similarly matches BloGDiT (53.86% versus 53.92%). The more severe GC100-10c→GC200-10c transfer exposes a limitation: BloGDiT reaches 18.58% while uTRM reaches 13.06%. This benchmark changes both graph size and local graph statistics, and therefore constitutes a distribution shift beyond pure size extrapolation (App. B.1).

On MIS, uTRM reaches an average independent-set size of 20.01, the strongest learned solution-free result in our comparison. It matches our SDDS re-implementation (20.00) at the same $17{,}280$ Transformer-layer evaluations and improves over CAM's (19.91). The oracle set size on this split is 20.12.

On Sudoku-Extreme, uTRM reaches 94.17% exact accuracy with best-of-100 inference (Tab. 1). To our knowledge, this is the first result above 90% on Sudoku-Extreme obtained from a model trained without solution labels. This comparison is not data- or compute-matched to supervised recursive reasoners: uTRM is trained on the full 3.83M-example training split without solution labels or additional augmentation and uses the verifier for best-of-K selection, whereas HRM and TRM are trained from 1K labeled base puzzles with extensive symmetry augmentation. Supervised methods with stochastic or large-scale multi-trajectory inference reach still higher accuracy, including PTRM (Sghaier et al., 2026) and Equilibrium Reasoners(Huang et al., 2026). In contrast to all our other uTRM models, we used $L_{\text{train}} = 8$ instead of $L_{\text{train}} = 2$, while H_{train} remains 2.

Across the three domains, the same solution-free training principle therefore matches or improves strong specialized unsupervised baselines, while exposing different benefits and limitations of recursive test-time computation. These comparisons do not isolate the contribution of recurrence itself, which we examine next under controlled architecture and compute matching.

4.3 Recurrence as a complementary test-time scaling axis

To determine what recursive computation contributes beyond terminal-verifier training itself, we first compare uTRM against flat Transformer policies optimized with the same objective.

At the training-time compute budget, recurrence does not provide an unconditional performance advantage. The difference emerges when inference depth is increased. Additional discrete refinement steps yield only small gains for the depth-matched Transformer, whereas repeatedly applying the shared recursive operator allows uTRM to overtake it. Thus, the main benefit of recurrence in this experiment is not improved performance at a fixed training depth, but the ability to reuse a compact learned operator for additional test-time computation.

Figure 2 makes this distinction explicit. On GC50, the flat Transformer initially performs better, but gains little beyond approximately $3\times$ its training-time depth. In contrast, uTRM continues to improve substantially beyond its training horizon and reaches its highest single-trajectory accuracy

Table 1: Main results across Graph Coloring, MIS, and Sudoku-Extreme. Scores are best-of-K unless stated otherwise; Graph Coloring and Sudoku report exact-solution accuracy (%), while MIS reports independent-set size. T_{eval} and H_{eval} denote discrete refinement steps and recursive cycles at inference, respectively, and layer evaluations include all K trajectories. For models trained on GC50, uTRM results are mean $\pm$ sample standard deviation over three training seeds; elsewhere, $\pm$ denotes three disjoint sampling replicates.

Task	Setting	Method	K	Disc. steps	H_{eval}	Layer evals.	Score ↑
Graph Coloring	5 colors, $n = 50$	ConsFormer	1	2300	–	9200	81.00
		BloGDiT	1	5399	–	21596	**83.00**
		uTRM	20	5	2	1200	81.33 ± 0.38
		uTRM	20	5	6	3600	**83.00 ± 0.00**
	5 colors, $n = 100^{\text{OOD}}$	ConsFormer	1	2310	–	9240	47.33
		ConsFormer	10	1634	–	65360	50.92
		ConsFormer	100	892	–	356800	55.67
		BloGDiT	1	5439	–	21756	54.67
		uTRM	20	5	2	1200	50.58 ± 1.92
		uTRM	20	5	6	3600	**56.64 ± 0.21**
	10 colors, $n = 100$	ConsFormer	1	?	–	NR	52.60
		BloGDiT	1	3728	–	26096	**53.92**
		uTRM	20	10	2	2400	53.78 ± 0.05
		uTRM	20	10	6	7200	53.86 ± 0.05
	10 colors, $n = 200^{\text{OOD}}$	ConsFormer	1	1490	–	10430	11.92
		BloGDiT	1	3931	–	27517	**18.58**
		uTRM	20	10	2	2400	12.42 ± 0.08
		uTRM	20	10	6	7200	13.06 ± 0.17
MIS	RB-small	OR-Tools (exact)	–	–	–	–	20.12
		LTFT	20	–	–	–	19.18
		DiffUCO	8	18	–	–	19.24
		SDDS (reported)	30	36	–	–	19.62
		CAM	20	50	–	–	19.91
		BloGDiT	1	1000	–	4000	19.52
		SDDS	30	12	–	4320	19.92 ± 0.00
		SDDS	30	36	–	12960	19.99 ± 0.00
		SDDS	30	48	–	17280	20.00 ± 0.01
		uTRM	30	12	2	4320	19.79 ± 0.01
		uTRM	30	12	6	12960	19.97 ± 0.00
		uTRM	30	24	4	17280	**20.01 ± 0.00**
Sudoku-Extreme	Supervised	HRM	1	–	–	–	55.00
		TRM	1	–	–	–	87.40
		PTRM	100	–	–	–	98.75
	Unsupervised	uTRM	1	50	2	1800	45.10
		uTRM	100	50	2	180000	94.17

^{OOD}Evaluated on harder or larger out-of-distribution instances.

at approximately $8\times$ the recursive depth used during training. The same qualitative effect is visible in mean coloring energy (App. Fig. 3), showing that the result is not solely an artifact of thresholding candidates into solved and unsolved instances.

The picture differs on MIS. Increasing recursive depth H alone improves uTRM, but is less effective than increasing the number of discrete refinement steps T. However, the two axes are complementary: jointly scaling H and T yields substantially stronger performance than scaling either axis alone and, at larger compute budgets, allows uTRM to match the SDDS Transformer baseline under matched inference compute despite using substantially fewer trainable parameters. For each compute budget, we select the allocation of H and T on the evaluation set and report its performance on the held-out test set. The preferred allocation of test-time compute is therefore task-dependent: Graph Coloring strongly favors additional latent recursion, whereas MIS benefits from combining latent recursion with additional discrete refinement. Parallel sampling is a third way to spend inference compute (App. C.3, Fig. 4). On GC50, additional recursion cycles help most at small and

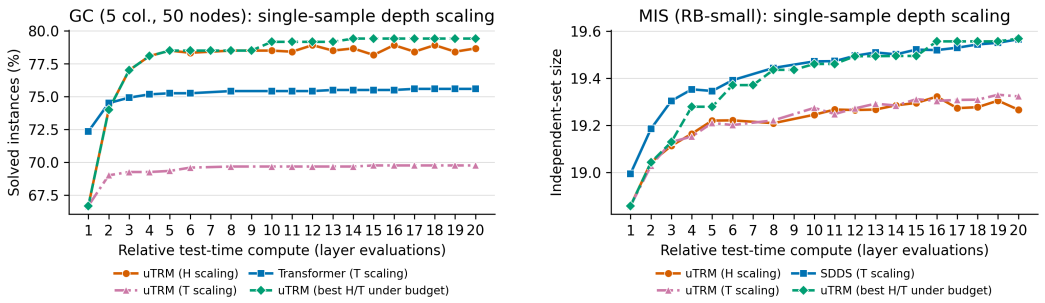

Figure 2: Single-sample test-time depth scaling on GC50 (left) and MIS RB-small (right). The x-axis denotes relative test-time compute $m = H_{\text{eval}}T_{\text{eval}}/(H_{\text{train}}T_{\text{train}})$. Curves vary recursive depth H, discrete refinement steps T, or select the best allocation of H and T under the same compute budget; the flat Transformer is scaled in T only.

moderate K, and all configurations converge as best-of-K accuracy approaches the benchmark ceiling. On MIS, the configurations remain close for all K.

4.4 What makes recursive depth extrapolatable?

A recursive reasoner should remain effective when its shared update operator is applied more often than during training. We therefore examine two factors that may determine this property: the recursive depth used during training and the gradient horizon through that recurrence.

App.C.5 Figure 5 shows that deeper recursive training does not automatically yield a more iterable operator. Under a fixed number of optimizer updates, the model trained with $H_{\text{train}} = 2$ already benefits strongly from extrapolation and reaches the highest accuracy in the sweep. The $H_{\text{train}} = 4$ model requires additional evaluation cycles to reach a similar level, while the model trained directly with $H_{\text{train}} = 8$ remains weaker over the evaluated range. Although this sweep is exploratory, it shows that performance at the training horizon and the usefulness of repeatedly applying the learned operator are not practically equivalent objectives.

Gradient truncation has an even clearer effect. TRM-style training propagates gradients only through the final high-level recursion cycle and earlier cycles are executed without gradient tracking. As shown in App. Fig. 3 (right), full backpropagation through both training cycles slightly improves performance at $H_{\text{eval}} = 2$, but yields almost no benefit from additional cycles. In contrast, detaching the first cycle produces a recursive operator whose accuracy continues to improve substantially as H_{eval} is increased.

Limitations. We do not provide results that allow directly comparing the efficiency of supervised versus unsupervised reasoners. Unlike the single-sample supervised baselines HRM and TRM, our best-of-K selection uses the energy at test time. Our instances have at most a few hundred variables, and larger benchmarks such as RB-large MIS remain untested.

5 Conclusion

Trained from terminal solution energy alone, recursive reasoners match or exceed specialized solution-free solvers in most settings and are, to our knowledge, the first solution-free models above 90% on Sudoku-Extreme. Latent recursion adds a test-time scaling axis distinct from refinement and parallel search, with a problem-dependent compute-optimal allocation. On Graph Coloring, repeatedly applying the shared operator lets uTRM overtake a depth-matched flat Transformer with about six times as many parameters, whereas on MIS it manages to close the gap by combining scaling axes. The Graph Coloring gains beyond the training depth arise only under truncated credit assignment, suggesting that solving at a fixed depth and learning a reusable operator are qualitatively distinct objectives. Learning to allocate recursion, refinement and parallel search per instance, and scaling solution-free recursive reasoners to larger instances, are natural next steps.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on proposing and refining hypotheses, research methodology, and experiments; to assist with implementation; to help interpret experimental results; to draft and revise parts of the manuscript; to search for related work; and to prepare submission materials such as keywords and the TL;DR.

We did not use generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, or provide critical ingredients for mathematical proofs. The paper does not contain formal proofs, so AI assistance with proof writing is not applicable.

All AI-assisted work was reviewed by the authors. AI-suggested research ideas and related-work leads were independently checked against the literature, and AI-assisted code was reviewed and tested for correctness by an author. We take full responsibility for the final content of this work, including any text, claims, code, or other artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

The main text and appendices specify the uTRM architecture (Sec. 3, App. B.3), verifier/energy functions (App.B.1), PPO implementation (App. B.2), training and inference configurations (Sec. 4.1), compute accounting (App. 3), dataset construction (App. B.4), and evaluation protocols(App. B.4). Public benchmark splits are used where available, and independently generated evaluation splits will be released. Upon acceptance, we will release the full training and evaluation code, experiment configurations, dataset-generation scripts, random seeds, and the additional datasets required to reproduce our experiments. We will additionally release checkpoints for the main uTRM and baseline models used in the reported comparisons and scaling experiments.

REFERENCES

Cem Anil, Ashwini Pokle, Kaiqu Liang, Johannes Treutlein, Yuhuai Wu, Shaojie Bai, J. Zico Kolter, and Roger B Grosse. Path independent equilibrium models can better exploit test-time computation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 7796–7809. Curran Associates, Inc., 2022. doi: 10.52202/068431-0566. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/331c41353b053683e17f7c88a797701d-Paper-Conference.pdf.

Junyeob Baek, Mingyu Jo, Minsu Kim, Mengye Ren, Yoshua Bengio, and Sungjin Ahn. Generative recursive reasoning. *arXiv preprint arXiv:2605.19376*, 2026. URL https://arxiv.org/abs/2605.19376.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20232–20242, 2022. URL https://arxiv.org/abs/2202.05826.

Alba Carballo-Castro, Julianna Piskorz, Paulius Rauba, Mihaela van der Schaar, and Pascal Frossard. Recursive scaling in masked diffusion models. *arXiv preprint arXiv:2606.18022*, 2026. URL https://arxiv.org/abs/2606.18022.

Shengyu Feng, Tarun Suresh, and Yiming Yang. Unsupervised diffusion solver for combinatorial optimization via combinatorial adjoint matching. In *Proceedings of the 43rd International Conference on Machine Learning*. PMLR, 2026. URL https://arxiv.org/abs/2605.30920.

Richard Freinschlag, Timo Bertram, Erich Kobler, Andreas Mayr, and Günter Klambauer. Symbol-equivariant recurrent reasoning models. In *Proceedings of the 43rd International Conference on Machine Learning*. PMLR, 2026. URL https://openreview.net/forum?id=1YCpxT3t5V.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 41340–41391. Curran Associates, Inc., 2025. doi: 10.52202/085713-1380. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Paper-Conference.pdf.

Benhao Huang, Zhengyang Geng, and J. Zico Kolter. Equilibrium reasoners: Learning attractors enables scalable reasoning. In *Proceedings of the 43rd International Conference on Machine Learning*. PMLR, 2026. URL https://arxiv.org/abs/2605.21488.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025. URL https://arxiv.org/abs/2510.04871.

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6659–6672. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/49f85a9ed090b20c8bed85a5923c669f-Paper.pdf.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43346–43367. PMLR, 2024. URL https://proceedings.mlr.press/v235/sanokowski24a.html.

Sebastian Sanokowski, Wilhelm Franz Berghammer, Haoyu Peter Wang, Martin Ennemoser, Sepp Hochreiter, and Sebastian Lehner. Scalable discrete diffusion samplers: Combinatorial optimization and statistical physics. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=peNgxpbdxB.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? Generalizing from easy to hard problems with recurrent networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 6695–6706, 2021. URL https://arxiv.org/abs/2106.04537.

Amin Sghaier, Ali Parviz, and Alexia Jolicoeur-Martineau. Probabilistic tiny recursive model. *arXiv preprint arXiv:2605.19943*, 2026. URL https://arxiv.org/abs/2605.19943.

Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers in Artificial Intelligence*, 3:580607, 2021. doi: 10.3389/frai.2020.580607. URL https://doi.org/10.3389/frai.2020.580607.

Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any CSP: Graph neural networks as fast global search heuristics for constraint satisfaction. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4280–4288. International Joint Conferences on Artificial Intelligence Organization, 2023. doi: 10.24963/ijcai.2023/476. URL https://doi.org/10.24963/ijcai.2023/476.

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. URL https://arxiv.org/abs/2506.21734.

Yudong W. Xu, Wenhao Li, Scott Sanner, and Elias B. Khalil. Self-supervised transformers as iterative solution improvers for constraint satisfaction. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 69432–69450. PMLR, 2025. URL https://proceedings.mlr.press/v267/xu25q.html.

Yudong W. Xu, Wenhao Li, Scott Sanner, and Elias B. Khalil. Large neighborhood search meets iterative neural constraint heuristics. *arXiv preprint arXiv:2603.20801*, 2026a. URL https://arxiv.org/abs/2603.20801.

Yudong W. Xu, Wenhao Li, Xiaoyu Wang, Scott Sanner, and Elias B. Khalil. Blocked Gibbs meets diffusion transformers: Unsupervised learning for constraint optimization. *arXiv preprint arXiv:2605.25129*, 2026b. URL https://arxiv.org/abs/2605.25129.

Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron C. Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with GFlowNets. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11952–11969, 2023. URL https://arxiv.org/abs/2305.17010.

A Appendix

A.1 Alternative diffusion-derived training objective

Our proposed model is trained using only the terminal objective in Eq. 10. To separate the effect of recursive architecture from the choice of solution-free training signal, we additionally train both recursive and non-recursive policies using the reverse-KL objective of SDDS (Sanokowski et al., 2025).

SDDS augments the terminal energy with a transition-level process reward derived from a prescribed forward corruption kernel q_I,

$$r_t^{\mathrm{rKL}} = \lambda_{\mathrm{rKL}} \left[\log q_I(x_t \mid x_{t-1}) - \log \pi_\theta(x_{t-1} \mid x_t) \right], \qquad t = T, \dots, 1, \tag{13}$$

together with the terminal energy reward. The first term rewards reverse transitions that are consistent with the prescribed forward process, while the second contributes the entropy of the learned refinement policy.

We use this objective only as a controlled alternative training signal. The proposed solution-free recursive reasoner does not require a forward process, and all our models use terminal verifier supervision unless stated otherwise.

B Experimental details

B.1 Definition of Energy Functions

MIS For Maximum Independent Set (MIS), let $x_v \in \{0, 1\}$ indicate whether vertex v is selected. We use

$$E_{\mathrm{MIS}}(\mathbf{x}) = -\alpha \sum_{v \in V} x_v + \beta \sum_{(u,v) \in E} x_u x_v,$$

where the first term rewards large sets and the second penalizes edges whose endpoints are both selected. We use $\alpha = 1$ and $\beta = 1.1$ by default.

Sudoku For Sudoku, let $x_i \in \{1, \dots, 9\}$ be the value of cell i. Let $\mathcal{U}$ contain all rows, columns, and 3×3 blocks, and let $n_{u,d}$ be the number of occurrences of digit d in unit u. Since the model can neither generate empty cells nor modify the given clues, the energy only counts duplicate digits:

$$E_{\mathrm{Sudoku}}(\mathbf{x}) = \sum_{u \in \mathcal{U}} \sum_{d=1}^{9} \max(n_{u,d} - 1, 0).$$

A valid Sudoku solution therefore has zero energy.

Graph Coloring For Graph Coloring, let $c_v \in \{0, \dots, K-1\}$ denote the color assigned to vertex v. Its energy is the number of monochromatic edges,

$$E_{\mathrm{GC}}(\mathbf{c}) = \sum_{(u,v) \in E} \mathbf{1}[c_u = c_v] + \sum_{v \in V} \mathbf{1}[c_v \notin \{0, \dots, K-1\}].$$

Consequently, a valid Sudoku solution or Graph Coloring has zero energy, while a valid MIS has energy equal to the negative size of the independent set.

B.2 PPO instantiation

We optimize the terminal verifier objective introduced in Sec. 3.2 using clipped PPO. In our main method, $\lambda_{\mathrm{rKL}} = 0$, such that PPO receives only the terminal verifier reward in Eq. 10. The corresponding ablations vary this reward coefficient while leaving the PPO implementation described below unchanged.

Table 2: Graph-coloring dataset statistics. Degree reports the pooled mean and standard deviation over graph vertices. Colorability intervals account for unresolved CP-SAT timeouts.

Setting	Split	Graphs	Nodes	Degree	Density	Colorable	Unresolved
GC-50/5	train	9000	50	6.88 ± 4.94	14.03%	7392/9000 (82.13%)	0
GC-50/5	test	1200	50	6.87 ± 4.93	14.03%	997/1200 (83.08%)	0
GC-50/5	held-out test	10000	50	7.75 ± 4.31	15.81%	5263/10000 (52.63%)	0
GC-100/5	test	1200	100	8.54 ± 4.72	8.63%	686/1200 (57.17%)	0
GC-100/10	train	9000	100	20.73 ± 7.58	20.94%	4855–4856/9000	1
GC-100/10	test	1200	100	20.71 ± 7.64	20.91%	647/1200 (53.92%)	0
GC-200/10	test	1200	200	33.70 ± 14.17	16.93%	248–603/1200	355

Reward and advantage normalization. Let d_I denote the number of decision variables in instance I. Recall that the rollout proceeds from x_T to x_0, such that refinement transition t maps x_t to x_{t-1}. We first rescale the reward from Sec. 3.2 by problem size,

$$\bar{r}_t = \frac{r_t}{d_I}. \tag{14}$$

For example, on GC50 the only nonzero raw reward is $-E_I(x_0)/50$ after the final refinement step.

Before computing advantages, we standardize $\bar{r}_t$ using exponential moving-average estimates of its mean and standard deviation,

$$\mu_k = \alpha\widehat{\mu}_k + (1-\alpha)\mu_{k-1}, \qquad \sigma_k = \alpha\widehat{\sigma}_k + (1-\alpha)\sigma_{k-1}, \tag{15}$$

where $\alpha = 0.05$ and $(\widehat{\mu}_k, \widehat{\sigma}_k)$ are computed jointly over all instances, sampled trajectories, and refinement steps in training iteration k. The reward used by PPO is

$$\widetilde{r}_t = \frac{\bar{r}_t - \mu_k}{\max(\sigma_k, 10^{-6})}. \tag{16}$$

This numerical normalization does not introduce any candidate-dependent intermediate supervision. In particular, we do not normalize rewards or returns separately per instance, per refinement step, or within the group of trajectories sampled for the same instance.

We use generalized advantage estimation with $\gamma = 1$, $\lambda = 0.95$, and zero terminal bootstrap value. Writing $V_t = V_\phi(I, x_t)$, transition t maps $x_t \to x_{t-1}$ and receives reward $\widetilde{r}_t$. We therefore compute

$$\delta_t = \widetilde{r}_t + \gamma V_{t-1} - V_t, \qquad A_t = \delta_t + \gamma\lambda A_{t-1}, \qquad V_0 = A_0 = 0, \qquad t = 1, \dots, T. \tag{17}$$

Here, the advantage recursion runs opposite to the rollout indexing: x_0 is terminal, while larger t corresponds to earlier states in the refinement trajectory. Advantages are standardized jointly over all trajectories and refinement steps, with a minimum standard deviation of 0.1, and clipped to $[-5, 5]$. Return targets are not otherwise normalized, and no group-relative baseline is used.

Value function. The value function shares the complete policy backbone. We mean-pool the final per-variable hidden representations and apply a two-hidden-layer MLP with widths 64 and 64, GELU activations, and a scalar output. Gradients from both the policy and value objectives update the shared backbone.

PPO ratio for factorized assignments. At refinement transition t, the policy samples the variables of x_{t-1} independently conditioned on (I, x_t). We therefore write the summed log-probability of the complete refinement action as

$$s_{\theta,t} = \sum_{i=1}^{d_I} \log \pi_\theta \left(x_{t-1,i} \mid I, x_t \right). \tag{18}$$

Since the summed log-probability scales with the number of variables, we form the PPO ratio from the mean log-probability difference,

$$\rho_t(\theta) = \exp\left(\frac{s_{\theta,t} - s_{\mathrm{old},t}}{d_I} \right). \tag{19}$$

Thus, ρ_t is the geometric mean of the per-variable probability ratios, rather than the joint probability ratio. We use the clipped surrogate

$$L_{\text{policy}} = -\mathbb{E}_t \left[\min \left(d_I \rho_t \widehat{A}_t,\ d_I \operatorname{clip}\left(\rho_t, 1-\epsilon, 1+\epsilon\right) \widehat{A}_t \right) \right], \qquad \epsilon = 0.02. \tag{20}$$

The factor d_I restores the first-order gradient scale of the summed log-probability objective away from the clipping boundary.

The value loss and combined objective are

$$L_{\text{value}} = \frac{1}{2}\mathbb{E}_t \left[\left(V_\phi(I, x_t) - R_t \right)^2 \right], \qquad L = 0.5 L_{\text{policy}} + 0.5 L_{\text{value}}. \tag{21}$$

We do not use value clipping, an entropy bonus, a KL penalty, or KL-based early stopping.

Rollout reuse and minibatching. Each training iteration collects fresh on-policy refinement trajectories and performs one shuffled minibatch pass over the resulting transitions; we do not perform multiple PPO epochs over the same rollout batch. For the GC50 experiments, we use $T = 5$ stochastic refinement steps, sample 20 trajectories per instance, and use optimizer minibatches of 320 transitions. We optimize with AdamW and global gradient-norm clipping at 1.16048.

B.3 uTRM Architecture Details

Our uTRM backbone uses a parameter-shared recurrent reasoning core with two Transformer layers, hidden dimension 128, eight attention heads, and SwiGLU feed-forward blocks with expansion factor 4. The same core is reused for all high- and low-level updates. In the graph-coloring configuration, each discrete refinement step performs $H = 2$ high-level cycles with $L = 2$ low-level updates per cycle. Tokens are represented by learned value embeddings and length-extensible sinusoidal positional encodings. The model additionally predicts their values through a categorical output head. MIS and Sudoku additionally condition the outer refinement policy on the normalized remaining refinement budget, while Graph Coloring does not require this conditioning.

Length-extensible latent initialization. At the beginning of every discrete refinement step, the high- and low-level states are initialized as

$$z^0_{H,i} = \bar{z}_H + r_H(i), \qquad z^0_{L,i} = \bar{z}_L + r_L(i),$$

where $\bar{z}_H, \bar{z}_L \in \mathbb{R}^{128}$ are learned vectors shared by all positions, while $r_H(i)$ and $r_L(i)$ are deterministic pseudorandom anchors generated from the position index i, using separate fixed seeds for the two latent levels. Consequently, initialization does not rely on a learned table with a fixed maximum length: anchors for additional positions can be generated on demand, allowing a model trained on smaller graphs to be applied to larger ones. Although the anchors are generated using a Gaussian pseudorandom function, they are fixed for each position and do not constitute run-time noise. The current candidate x_t is represented by learned value embeddings $\mathbf{x}_t$, which are re-injected at every low-level update according to 2.

Discrete breaks in the latent chain. The recurrent states are propagated only within a single refinement step. After the model produces logits, the next discrete assignment x_{t-1} is sampled from the factorized categorical policy conditioned on x_t. At the following step, both latent states are reinitialized and x_{t-1} enters the recurrence as described in Eq. (2). Neither z_H nor z_L is carried across this boundary. Thus, in contrast to the continuous latent chain of TRM, information between successive refinement steps must pass through the discrete assignment:

$$(z^0_H, z^0_L, x_t) \longrightarrow (z^\star_H, z^\star_L) \longrightarrow x_{t-1} \longrightarrow (z^0_H, z^0_L, x_{t-1}).$$

Constraint-aware attention. Task constraints enter the reasoning core through pairwise attention relations. For Graph Coloring, a hard mask permits a vertex to attend only to itself and to adjacent vertices,

$$M_{ij} = \begin{cases} 0, & i = j \text{ or } (i,j) \in E, \\ -\infty, & \text{otherwise.} \end{cases}$$

MIS uses the same adjacency-based mask, since its pairwise conflicts are likewise defined by graph edges. Sudoku instead supplies three relation channels indicating whether two cells share a row, column, or box. These channels are transformed into learned, head-specific additive attention biases, they do not hard-mask unrelated cells.

Table 4: uTRM PPO training hyperparameters. The learning rate follows a cosine schedule from its initial to its target value. All runs use AdamW with weight decay 10^{-4} and $(\beta_1, \beta_2) = (0.9, 0.999)$. Each epoch rolls out K trajectories for each of N training instances; the resulting NKT_{train} transitions form the update batch, which is consumed in one shuffled pass of PPO minibatches. Updates count the optimizer steps performed during training. L is the number of L-cycles per H-cycle.

Task	Learning rate	Updates	N	K	Update batch	Minibatch	$(T_{\text{train}}, H_{\text{train}}, L)$
GC-5c-50n	$2.9\times10^{-5} \to 9.7\times10^{-6}$	619k	9,000	20	900k	320	$(5, 2, 2)$
GC-10c-100n	$2.9\times10^{-5} \to 9.7\times10^{-6}$	2.81M	9,000	20	1.8M	320	$(10, 2, 2)$
MIS-small	$1.8\times10^{-5} \to 3\times10^{-7}$	1.17M	4,000	5	240k	256	$(12, 2, 2)$
Sudoku-Extreme	$3.0\times10^{-5} \to 7.5\times10^{-6}$	9.7M	9,000	32	4.61M	171	$(16, 2, 8)$

B.4 Data and Evaluation Protocol

Graph Coloring. We use the graph-coloring datasets of ConsFormer (Xu et al., 2025) and follow their evaluation protocol. Since no separate validation split is provided for the in-distribution GC50 benchmark, we evaluate checkpoints directly on the released test split and use this split for model selection. We therefore treat the corresponding GC50 results as test-set-selected. Generalization is evaluated separately on the larger out-of-distribution graphs, which are not used for model selection.

Maximum independent set. For MIS, we generate training and validation graphs independently following the data-generation procedure of SDDS (Sanokowski et al., 2025). All model and checkpoint selection is performed exclusively on this self-generated validation set. After fixing the model configuration, we evaluate once on the exact released CAM (Feng et al., 2026) test set used for the reported results in Table 1. Thus, unlike for GC50, the reported MIS test split is not used during model selection. To ensure feasibility, both our SDDS reimplementation and uTRM employ Conditional Expectation in the final decoding step.

Exact MIS baseline. For comparison with an exact solver, we evaluate OR-Tools CP-SAT 9.15.6755 directly on the same 1,000-instance CAM test split as the neural methods. We use eight search workers and a 10-second per-instance time limit. OR-Tools certifies all 1,000 instances as optimal; the slowest instance requires 0.663 seconds, so the time limit is never active. The resulting mean maximum-independent-set size is 20.115, reported as 20.12 at the precision used in Table 1.

Sudoku-Extreme. We train on the full Sudoku-Extreme training split released with HRM (Wang et al., 2025) (subsample $9,000$ puzzles per epoch) and use a fixed subset of $1,000$ test instances for evaluation and checkpoint selection during training. Final results are computed on the remaining samples from the 422,786-instance test split.

B.5 Training details

Table 3: Model size and training duration for our learned methods. Parameter counts include the policy and value head. Training time is measured in wall-clock hours and averaged across training seeds when multiple seeds are available. All models were trained on a single NVIDIA H200 with 141GB memory.

Training data	Model	Trainable parameters	Training time (h)
GC-5c-50n	uTRM	540,616	1.89
GC-10c-100n	uTRM	541,901	19.83
MIS-small	uTRM	539,845	67.43
MIS-small	SDDS	2,524,869	63.29
Sudoku-Extreme	uTRM	541,668	40.69

Table 5: Ablation of diffusion-derived process rewards on GC50. Best-of-20 metrics are computed after selecting the lowest-energy sample among 20 trajectories, while mean-sample metrics average over all trajectories. Values are mean ± sample standard deviation over three training seeds, evaluated at the training-time inference configuration.

Model	Process rewards	Accuracy (%) ↑		Energy ↓	
		Best-of-20	Mean sample	Best-of-20	Mean sample
Transformer	No	82.22 ± 0.21	74.65 ± 1.27	0.243 ± 0.008	0.431 ± 0.030
Transformer	Yes	79.97 ± 3.88	70.66 ± 7.44	0.284 ± 0.081	0.527 ± 0.201
uTRM	No	81.33 ± 0.38	68.23 ± 1.58	0.265 ± 0.010	0.605 ± 0.037
uTRM	Yes	81.86 ± 0.38	69.01 ± 0.66	0.253 ± 0.008	0.564 ± 0.009

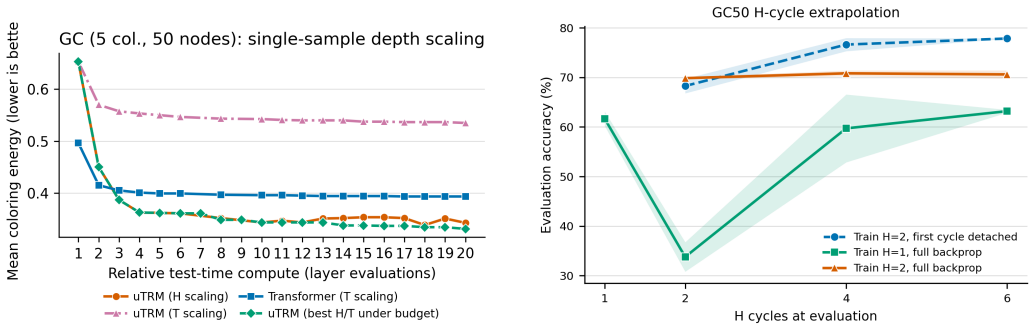

Figure 3: Conditions for recursive test-time depth extrapolation on GC50. **Left**: mean coloring energy under single-sample test-time scaling by recursive depth H, discrete refinement depth T, or the best H/T allocation under a fixed compute budget. **Right**: evaluation accuracy as H_{eval} increases for models trained with different recursive depths and gradient horizons.

C Additional Experiments

C.1 Diffusion-Process Rewards

We test whether the diffusion-derived process rewards of the reverse-KL objective are necessary for solution-free training by comparing the full objective against terminal-energy supervision alone. Removing the process rewards does not reduce solution quality: for the Transformer it slightly improves both accuracy and energy on average, while for uTRM the two objectives perform similarly. Thus, in this setting, matching the prescribed diffusion process provides no clear advantage over directly optimizing the quality of the terminal solution. This is consistent with the mismatch between standard global denoising and discrete constraint optimization observed by Xu et al. (2026b), suggesting that an effective optimization trajectory need not coincide with a prescribed reverse-diffusion process.

C.2 Graph-Coloring Energy along Depth-Scaling

The left panel in fig. 3shows that increasing recursive depth continues to reduce terminal energy, while increasing only the number of discrete refinement steps saturates earlier. The right panel of fig. 3 isolates the effect of the training credit horizon: with $H_{\text{train}} = 2$, truncating gradients through the first high-level cycle enables substantial gains from increasing H_{eval}, whereas full backpropagation largely removes this extrapolation benefit. Training with only one high-level cycle also fails to extrapolate reliably to deeper evaluation.

C.3 Parallel Scaling

Figure 4 shows how increasing the number of independent samples interacts with recursive depth H and discrete refinement depth T. On GC50, configurations with larger recursive depth achieve

Table 6: Recurrence versus flat Transformers on GC50. Values are best-of-20 accuracy (%) over three training seeds.

Model	Matching criterion	Test (raw)		Held-out (solvable)	
		Training	3×	Training	3×
uTRM (2 shared layers)	reference	81.33 ± 0.38	83.00 ± 0.00	97.20 ± 0.31	99.62 ± 0.09
Transformer (2 layers)	parameter-matched	67.31 ± 0.86	74.92 ± 1.75	73.45 ± 1.29	85.81 ± 1.46
Transformer (12 layers)	depth-matched	82.22 ± 0.21	82.47 ± 0.29	98.73 ± 0.48	98.99 ± 0.36

For uTRM, 3× compute increases H-cycles from 2 to 6. For flat Transformers, it increases discrete refinement steps from 5 to 15. The 2-layer and 12-layer flat models match the recurrent model in trainable parameter count and effective Transformer depth, respectively. Test accuracy is unadjusted. Held-out accuracy divides solved instances by the 5263 proven-solvable instances out of 10000.

substantially better single-sample performance, while the gap narrows as K increases. Increasing T improves the shallow-recursion configuration, but the deeper-recursion model remains strongest over most of the evaluated range. At large K, all configurations approach similar best-of-K accuracy, indicating that parallel sampling can compensate for differences in individual trajectory quality.

On MIS RB-small, the compared configurations are much closer across all values of K. All methods improve consistently with additional parallel samples, while differences due to H and T remain comparatively small. The best-of-K curves increasingly converge as K grows, showing that the benefit of stronger single-trajectory configurations diminishes under larger sampling budgets.

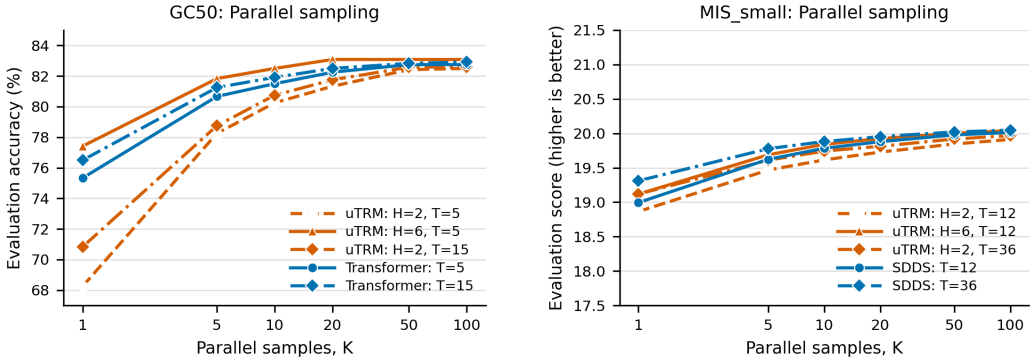

Figure 4: Best-of-K parallel-sampling scaling on GC50 (left) and MIS RB-small (right). We vary recursive depth H, discrete refinement depth T, and the number of independent trajectories K. Reported scores are the best solution obtained among K samples.

C.4 Graph-Coloring 50 Held-out test set

To verify that the GC50 comparison is not an artifact of the benchmark split used during model selection, we additionally evaluate on 10,000 independently generated graphs drawn from a similar distribution as the test split (slightly denser), of which 5,263 are proven 5-colorable. We therefore report accuracy over the solvable subset. At training-time compute, the depth-matched Transformer slightly outperforms uTRM (98.73% vs. 97.20%), but with 3× inference compute uTRM improves to 99.62%, compared with 98.99% for the Transformer, the parameter-matched baseline remains substantially weaker. Thus, the held-out evaluation reproduces the main qualitative result: untied depth is stronger at the training horizon, whereas recursive weight sharing benefits more from additional inference-time computation.

C.5 H-extrapolability for different H_{train}

Figure 5 shows the full H-cycle scaling results for models trained at different recursive depths. On GC50, models trained with $H_{\text{train}} \in \{2, 4\}$ converge to approximately the same performance as

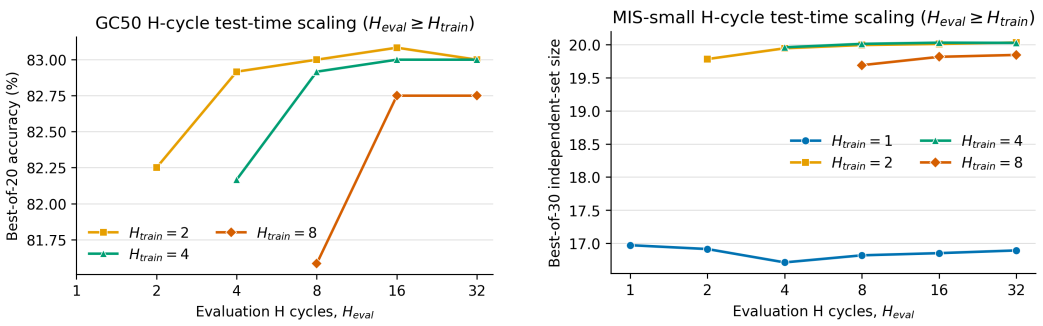

Figure 5: H-cycle test-time scaling for uTRMs trained at different recursive depths H_{train}. We evaluate only $H_{\text{eval}} \geq H_{\text{train}}$. Left: GC50 best-of-20 solution accuracy. Right: MIS-small best-of-30 independent-set size.

H_{eval} is increased, reaching about 83% best-of-20 accuracy. The model trained with $H_{\text{train}} = 8$ benefits substantially from additional evaluation cycles, but remains below the shallower-trained models over the evaluated range. A similar pattern appears on MIS-small: $H_{\text{train}} \in \{2, 4\}$ yield the strongest results and converge to approximately 20 best-of-30 independent-set size, while the $H_{\text{train}} = 8$ model improves with additional cycles but remains lower. The $H_{\text{train}} = 1$ MIS model performs substantially worse and does not recover through test-time depth scaling.