Deep Improvement Supervision

Arip Asadulaev [1]   **Rayan Banerjee** [1]   **Fakhri Karray** [1]   **Martin Takac** [1]

Abstract

Recently, it was shown that small, looped architectures, such as Tiny Recursive Models (TRMs), can outperform Large Language Models (LLMs) on complex reasoning tasks, including the Abstraction and Reasoning Corpus (ARC). In this work, we investigate a core question: how can we further improve the efficiency of these methods with minimal changes? To address this, we frame the latent reasoning of TRMs as a form of classifier-free guidance and implicit policy improvement algorithm. Building on these insights, we propose a novel training scheme that provides a target for each loop during training. We demonstrate that our approach significantly enhances training efficiency. Our method reduces the total number of forward passes by 18× and eliminates halting mechanisms, while maintaining quality comparable to standard TRMs. Notably, we achieve 24% accuracy on ARC-1 with only 0.8M parameters, outperforming most LLMs.

1. Introduction

The goal of reasoning models is to start from a set of specific examples or observations and infer a general rule or pattern. Such models systematically manipulate and connect pieces of information to infer new conclusions and solve problems not explicitly stated in the training data. Building such models is a key challenge for current machine learning approaches. The use of iterative refinement loops on a model's outputs has become a major driver of progress in deep learning reasoning, a trend evident in both small and large-scale architectures.

At a high level, the reasoning process in large language models (LLMs) can be viewed as a product of this same principle: a recursive, loop-like refinement of the model's own outputs (Hurst et al., 2024; Jaech et al., 2024). Their primary reasoning mechanism, Chain-of-Thought (CoT) prompting (Wei et al., 2022), relies on externalizing reason-
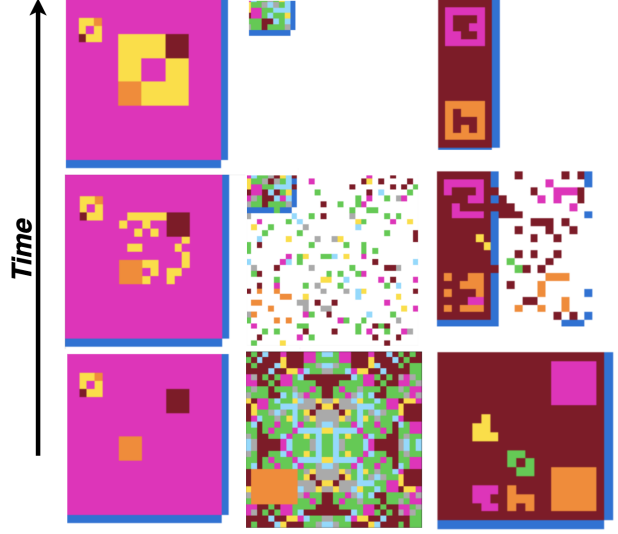

Figure 1. Blueprint of the discrete diffusion process on the ARC. Starting from the input x, following timestep t, we generate diffusion steps to the target y (Chollet, 2019).

ing into a sequence of generated text steps. However, this process in LLMs demands extensive computational infrastructure, making it highly expensive.

For smaller models, this idea is also proved itself, looped transformers (Giannou et al., 2023; Yang et al., 2023) repeatedly apply the same transformer block with input injection at each step and achieve better performance than a standard one forward pass transformer on reasoning and meta learning tasks, why using 10x less number of parameters (Yang et al., 2023).

Recently, models like Hierarchical Reasoning Models (HRM) (Wang et al., 2025) and Tiny Recursive Models (TRM) (Jolicoeur-Martineau, 2025), was built on the similar idea of reusing model output repeatedly in both the input and latent spaces. These models have demonstrated very impressive performance and even outperformed multi-billion parameter LLM models on complicated ARC-AGI (Chollet, 2019) tasks. However, a path of building reasoning models based on looped inference introduces a profound question. When a reasoning model executes a refinement step, what guaranties that it is on a path to a better answer? *How can we guide the model to ensure that each step is verifiably closer to the right conclusion?*

[1]MBZUAI. Correspondence to: Arip Asadulaev <arip.asadulaev@mbzuai.ac.ae>.

In this paper, we propose a learning algorithm that makes the reasoning process more task-oriented. Our main finding is that guiding the model to predict intermediate steps via discrete diffusion (see Fig. 1) during iterative reasoning significantly reduces training complexity and improves generalization. We demonstrate that, with a target for each supervision step, the `no-grad` cycles proposed for TRM and HRM can be avoided. Additionally, by performing a fixed number of refinement steps, we simplify training because the need to halt is eliminated.

To formalize our approach and hypotheses regarding the efficacy of TRMs, we first model the latent reasoning process through distinct theoretical lenses, establishing a tractable and well-defined formulation. Specifically, we show that a TRM can be interpreted as a form of classifier-free diffusion guidance (Sec. 3.1). Furthermore, we demonstrate how asymmetric latent reasoning facilitates policy improvement that exceeds the policy inherent in the training data (Frans et al., 2025) (Sec. 3.2). Finally, based on this formulation, we theoretically justify the benefits of step-wise guidance.

Our method achieves state-of-the-art performance on complex reasoning benchmarks, including `ARC-AGI 1`, `ARC-AGI 2`, using a much simpler architecture than the TRM. We avoid training the halting step module, use 3x fewer supervision steps and 8x fewer latent reasoning steps. As a highlight of our method, our model achieves a 24% accuracy on `ARC-AGI-1` outperform most of the existing open source LLM models without any external knowledge.

2. Background

2.1. Hierarchical Reasoning Models

A looped (Giannou et al., 2023) and universal (Dehghani et al., 2018) transformers repeatedly applies the same transformer block, with input injection each time, and is trained to make its intermediate loop output correct. This model was applied for the various tasks, showing the improvement over the single step models (Yang et al., 2023). Based on this idea, Hierarchical Reasoning Models (HRMs) (Wang et al., 2025) are supervised sequence-to-sequence models that perform *recursive refinement* of a prediction by interleaving two small recurrent networks that operate at different update frequencies. Let $\tilde{\mathbf{x}} \in \mathcal{V}^L$ denote an input sequence of length L on a vocabulary $\mathcal{V}$, and let $\mathbf{y} \in \mathcal{V}^L$ be the desired output. HRM uses an input embedding f_I, two recurrent reasoning modules a *low-level* module f_L and a *high-level* module f_H and an output head f_O. After embedding $\mathbf{x} = f_I(\tilde{\mathbf{x}}) \in \mathbb{R}^{L \times D}$, HRM carries two latent states $\mathbf{z}_L, \mathbf{z}_H \in \mathbb{R}^{L \times D}$ through the supervision steps. Within a forward pass, it performs n updates of f_L^ϕ for every update of f_H^ψ and repeats this T times before decoding with f_O. A typical schedule used in previous work is $n = 2$ and $T = 2$.

The final prediction is $\hat{\mathbf{y}} = \arg\max f_O(\mathbf{z}_H)$. During a forward pass, HRM evaluates the following updates:

$$\mathbf{z}_L^{n+1} \leftarrow f_L^\phi((\mathbf{z}_L^n + \mathbf{x}) + \mathbf{z}_H^n), \quad \text{(repeated } n \text{ times)} \quad (1)$$
$$\mathbf{z}_H^{n+1} \leftarrow f_H^\psi(\mathbf{z}_L^{n+1} + \mathbf{z}_H^n),$$
$$\hat{\mathbf{y}} = \arg\max f_O(\mathbf{z}_H).$$

Most evaluations in the early part of the schedule are executed without gradient tracking, while the final evaluations are backpropagated through. This design aims to amortize compute while allowing the model to refine internal states before a gradient-bearing step.

Deep supervision. To emulate very deep computation without prohibitive memory, HRM reuses $(\mathbf{z}_L, \mathbf{z}_H)$ across N_{sup} supervision steps up to 16, detaching the states between steps. This *deep supervision* improves the answer iteratively and yields hundreds of effective layers, while avoiding full backpropagation over time.

Adaptive Computational Time. Training-time efficiency is improved by a learned halting mechanism (ACT). A small head predicts whether to stop iterating on the current example or continue; the published implementation trains this with a halting loss and an additional continue loss that requires an *extra* forward pass, effectively doubling forward compute per optimization step (Graves, 2016). The test-time evaluation runs a fixed maximum number of supervision steps to maximize accuracy.

2.2. Tiny Recursive Models

Tiny Recursive Models (Jolicoeur-Martineau, 2025) retain the core idea of iterative refinement but collapse HRM's complexity into a *single* tiny network and a simpler recursion scheme. In the TRM setup, $\mathbf{z}_H$ is the state that the model reads out to produce the answer (the output head is applied to $\mathbf{z}_H$: $\hat{\mathbf{y}} = \arg\max f_O(\mathbf{z}_H)$). $\mathbf{z}_L$ is a *working memory* state that's updated using the input $\mathbf{x}$ and the current answer, and is then used to update $\mathbf{z}_H$. Because the loss is applied on the prediction from $\mathbf{z}_H$, optimization pressure makes $\mathbf{z}_H$ look like (encode) the current solution. On the other hand, $\mathbf{z}_L$ is only supervised indirectly through its effect on $\mathbf{z}_H$, so it is free to be an internal reasoning representation rather than a decodable solution. Within same f^ϕ network for L and H module, TRM repeats each recursion equal to HRM Eq. 1.

The prediction is made from $\mathbf{z}_H$ through the output head and trained with cross-entropy. This asymmetry (only $\mathbf{z}_H$ sees final $\mathbf{z}_L^{n+1}$; and only $\mathbf{z}_H$ is decoded and penalized) naturally pushes $\mathbf{z}_H$ towards the space of valid answers, while $\mathbf{z}_L$ becomes the latent *reasoning scratchpad* that helps improve the next $\mathbf{z}_H$. The TRM paper explicitly reframes this: "$\mathbf{z}_H$ is simply the current (embedded) solution... $\mathbf{z}_L$ is a latent feature that does not directly correspond to a solution but

can be transformed into one by f_H. It was show on the Sudoku example that when you "reverse-embed + argmax," the tokenized $\mathbf{z}_H$ looks like the solved grid, while tokenized $\mathbf{z}_L$ is not.

From the notation perspective, we want to note that TRM renames $\mathbf{z}_H$ to $\mathbf{y}$ (the running answer) and $\mathbf{z}_L$ to $\mathbf{z}$ (latent reasoning). The loop becomes: update $\mathbf{z}$ using $(\mathbf{x}, \mathbf{y}, \mathbf{z})$; then update $\mathbf{y}$ using $(\mathbf{y}, \mathbf{z})$. Carrying both across deep-supervision steps lets the model iterate: $\mathbf{z}$ remembers how it got to the current guess (like a chain-of-thought), and $\mathbf{y}$ stores the current guess itself. TRM trains a *single* halting probability via binary cross-entropy against correctness. Let $\mathcal{L}_{\text{task}} = \text{CE}(f_O(\mathbf{y}), \mathbf{y}_{\text{true}})$ be the prediction cross-entropy loss and $\mathcal{L}_{\text{halt}} = \text{BCE}(q(\mathbf{y}), \hat{\mathbf{y}} = \mathbf{y}_{\text{true}})$ the halting loss with $q(\cdot)$ a scalar head. An optimization step iterates up to N_{sup} supervision steps, performing $T - 1$ no-grad recursion cycles, then one with gradients, detaching $(\mathbf{y}, \mathbf{z})$ between supervision steps. Early stopping within a minibatch is permitted by using the halting signal.

2.3. Diffusion Guidance as Policy Improvement

We use the policy improvement point of view to describe the latent reasoning process of TRM. For clarity, we provide a short background section below.

Recent work formalizes a tight connection between *classifier-free guidance* (CFG) in diffusion/flow models and *policy improvement* in reinforcement learning (RL) (Frans et al., 2025). This work provides a framework for the analysis of diffusion models as RL methods. For a state $\mathbf{s}$ and an action/output $\mathbf{y}$, the core construction, termed classifier-free guidance RL (CFGRL), parameterizes a target policy as:

$$\pi(y \mid \mathbf{s}) \propto \hat{\pi}(y \mid \mathbf{s}) f\big(A_{\hat{\pi}}(\mathbf{s}, \mathbf{y})\big), \qquad (2)$$

where $f : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a nonnegative monotonically increasing function of the advantage $A^\pi(\mathbf{s}, \mathbf{y}) = Q^\pi(\mathbf{s}, \mathbf{y}) - V^\pi(\mathbf{s})$ (Sutton et al., 1998), and $\hat{\pi}$ is a reference policy. According to the formal proof of the authors in the appendix (Theorem 1) (Frans et al., 2025), a new policy generated from a carefully weighted combination of previous policies is guaranteed to achieve a higher expected return than the original reference policy. This result generalizes earlier findings in bandits and provides a clear pathway for improving policies.

We can say that the reference policy is learned from given data, or we can say that this is any non-optimal policy that we want to improve. Following (2) π provably improves over $\hat{\pi}$; moreover, *attenuating* the optimality factor with an exponent w yields a family $\pi_w \propto \hat{\pi} f(A)^w$ whose expected return increases with w (up to the distribution-shift limits). This recovers KL-regularized policy improvement as a special case, where the optimal solution has the same product form with $f(A) = \exp(A/\beta)$, tying w to the trust-region

strength β *without* retraining.

Crucially, CFGRL shows that diffusion guidance composes these factors at the *score* level. Let $\mathbf{o} \in \{\emptyset, 1\}$ indicate *optimality*, defined by f as $p(\mathbf{o}|\mathbf{s}, \mathbf{y}) = f(A(s, \mathbf{y}))$, then the product policy of Eq. (2) can now be equivalently defined as: $\pi(y \mid \mathbf{s}) \propto \hat{\pi}(y \mid \mathbf{s}) p(\mathbf{o}|\mathbf{s}, \mathbf{y})$.

As such, the score of the product policy above can be represented as the sum of two factors $\nabla_a \log \pi(y|\mathbf{s}) = \nabla_a \log \hat{\pi}(y|\mathbf{s}) + \nabla_a \log p(\mathbf{o}|\mathbf{s}, y)$. Then, using Bayes rule and w, CFG produces an score:

$$\nabla_y \log \pi_w(y \mid \mathbf{s}) = \nabla_y \log \hat{\pi}(y \mid \mathbf{s}) +$$

$$w\Big(\nabla_y \log \hat{\pi}(y \mid \mathbf{s}, \mathbf{o}=1) - \nabla_y \log \hat{\pi}(y \mid \mathbf{s})\Big), \qquad (3)$$

which corresponds to sampling from $\pi_w \propto \hat{\pi}(y \mid \mathbf{s}) p(\mathbf{o}=1 \mid s, \mathbf{y})^w$. Please see (Frans et al., 2025) for other derivation details. This realizes *controllable* policy improvement at test time by tuning w, while training remains a simple supervised diffusion/flow objective with classifier-free dropout of $\mathbf{o}$. Overall, CFGRL preserves the stability and simplicity of generative supervised training while providing a principled, test-time knob for step-size in policy improvement. Empirically, CFGRL shows policy improvement and yields consistently better returns than the reference policy $\hat{\pi}$

3. Different Faces of Latent Reasoning

In this section, we decompose the architectural elements of the TRM to demonstrate its relationship with well-established frameworks in generative modeling and reinforcement learning. The goal is to provide a formal hypothesis for *why* TRM works by connecting the model's asymmetric latent updates to **Classifier-Free Guidance (CFG)** and to **Implicit Policy Improvement**.

Notation. Let the state be $\mathbf{s} = (\mathbf{x}, \mathbf{z}_L, \mathbf{z}_H)$ at recursion step t, where $\mathbf{x}$ is the input, $\mathbf{z}_L$ is the latent reasoning *scratchpad*, and $\mathbf{z}_H$ is the current solution embedding. TRM applies a (parameter-shared) update operator in two roles:

$$\mathbf{z}_L^{n+1} = f_\phi^{(L)}(\mathbf{z}_L^n, \mathbf{z}_H^n, \mathbf{x}), \qquad \mathbf{z}_H^{n+1} = f_\phi^{(H)}(\mathbf{z}_H^n, \mathbf{z}_L^{n+1}), \tag{4}$$

where $f_\phi^{(L)}$ and $f_\phi^{(H)}$ are two instantiations that *share* parameters ϕ but differ in inputs. An output head f_O maps $\mathbf{z}_H$ to a vector of logits. As established in §2.2, only $\mathbf{z}_H$ is decoded and trained (with cross-entropy).

We introduce a binary indicator $o \in \{0, 1\}$ to distinguish two readout modes:

- **Reference** ($o = 0$): pre-reasoning readout from $\mathbf{z}_H^n$, with $\ell_u := f_O(\mathbf{z}_H^n)$ is *reference* logits.

- **Conditional** ($o = 1$): post-reasoning readout from $\mathbf{z}_H^{n+1}$ after attending to $\mathbf{z}_L^{n+1}$, with $\ell_c := f_O(\mathbf{z}_H^{n+1})$ is *conditional* logits).

Throughout, we write $\pi_u = \text{softmax}(\ell_u)$, $\pi_c = \text{softmax}(\ell_c)$. Logits are understood as unnormalized log-probabilities whose softmax defines the corresponding policy over actions/classes.

3.1. Latent Reasoning as Classifier-Free Guidance

The Hierarchical Reasoning Model and the Tiny Recursive Model are based on iterative latent updates. As discussed in §2.1 and §2.2, the process is asymmetric: the reasoning state $\mathbf{z}_L$ is updated using $\mathbf{x}$ and $\mathbf{z}_H$, and this updated reasoning conditions the subsequent update of $\mathbf{z}_H$. At any recursion step t, this asymmetry yields two readouts:

$$\ell_u = f_O(\mathbf{z}_H^n)(\text{reference}), \qquad \ell_c = f_O(\mathbf{z}_H^{n+1})(\text{conditional}).$$

By interpolating in *logit* space we obtain guided logits and the corresponding guided policy:

$$\ell_w = (1-w)\,\ell_u + w\,\ell_c, \qquad \pi_w(\cdot \mid \mathbf{s}) = \text{softmax}(\ell_w). \tag{5}$$

Equivalently, π_w is a product-of-experts blend of the two policies:

$$\pi_w(a \mid \mathbf{s}) \propto \pi_u(a \mid \mathbf{s})^{1-w}\,\pi_c(a \mid \mathbf{s})^w.$$

Here, the residual $\Delta\ell := \ell_c - \ell_u$ captures the *direction of improvement* suggested by latent reasoning, while w acts as a guidance scale (or reasoning depth).

Practical note. *In standard TRM inference, decoding uses the conditional path $\mathbf{z}_H^{n+1}$ (i.e., $w = 1$). Eq. (5) exposes w as a controllable knob at test time, allowing amplification or attenuation of the reasoning signal.*[1]

3.2. Latent Reasoning as Implicit Policy Improvement

Section 3.1 shows that TRM's two readouts can be mixed as in CFG. We now show that the same mechanism implements a controllable *policy improvement* operator. Consider sampling from a product policy:

$$\pi_{\text{new}}(y \mid \mathbf{s}) \propto \underbrace{\hat{\pi}(y \mid \mathbf{s})}_{\text{Reference policy}} \cdot \underbrace{p(o = 1 \mid \mathbf{s}, y)}_{\text{Optimality likelihood}}, \tag{6}$$

where $p(o = 1 \mid \mathbf{s}, y)$ measures how likely action y leads to an optimal outcome (or, in our approximation below, to an improved conditional distribution).

---

[1]Classifier-Free Guidance in diffusion mixes *scores* (gradients of log density). Here we mix *logits*. The algebraic product-of-experts identity above holds because logits exponentiate to probabilities under softmax.

Guidance-based RL (Frans et al., 2025) shows that such product policies improve performance under mild conditions: if $p(o = 1 \mid \mathbf{s}, y)$ increases monotonically with the advantage $A_{\hat{\pi}}(\mathbf{s}, y)$, then the product policy has higher expected return than $\hat{\pi}$. Introducing a guidance scale $w \geq 0$ and applying Bayes' rule,

$$\log p(o \mid \mathbf{s}, y) = \log \hat{\pi}(y \mid \mathbf{s}, o) - \log \hat{\pi}(y \mid \mathbf{s}) + \log p(o \mid \mathbf{s}),$$

where $\log p(o \mid \mathbf{s})$ is independent of y, we obtain

$$\log \pi_{\text{new}}(y \mid \mathbf{s}) \propto \log \hat{\pi}(y \mid \mathbf{s}) +$$
$$w\big(\log \hat{\pi}(y \mid \mathbf{s}, o) - \log \hat{\pi}(y \mid \mathbf{s})\big)$$
$$\equiv (1-w)\underbrace{\log \hat{\pi}(y \mid \mathbf{s})}_{\text{"old"}} + w\underbrace{\log \hat{\pi}(y \mid \mathbf{s}, o)}_{\text{"new"}}.$$

Exponentiating and normalizing yields the exact identity

$$\pi_w(y \mid \mathbf{s}) \propto \hat{\pi}(y \mid \mathbf{s}) \cdot \left(\frac{\hat{\pi}(y \mid \mathbf{s}, o = 1)}{\hat{\pi}(y \mid \mathbf{s})}\right)^w \tag{7}$$

which matches the logit interpolation in Eq. (5) when ℓ_u and ℓ_c are the (unnormalized) log-probabilities of $\hat{\pi}(\cdot \mid \mathbf{s})$ and $\hat{\pi}(\cdot \mid \mathbf{s}, o = 1)$, respectively.

By training the conditional path ($\mathbf{z}_H^{n+1}$, hence ℓ_c) toward the correct answer while the reference path ($\mathbf{z}_H^n$, hence ℓ_u) encodes the pre-reasoning baseline, the residual $\Delta\ell = \ell_c - \ell_u$ functions as a log-likelihood ratio—an advantage-like update that reweights actions the reasoning step deems *better*. Thus TRM performs a step of policy iteration at each forward pass without explicitly estimating a value function.

4. Deep Improvement Supervision

While §3.2 establishes that TRMs implicitly perform policy improvement, relying solely on terminal supervision creates a difficult optimization landscape: the model must discover a sequence of latent updates that yields a correct final answer, inferring the direction of improvement from a sparse, high-variance terminal signal. To address this, we introduce **Deep Improvement Supervision (DIS)**. We first formalize the condition under which the asymmetric readouts improve predictions—*Advantage Margin*—and then show how DIS directly optimizes this condition by turning "reasoning" into a concrete, step-wise supervised objective.

4.1. The Mechanics of Improvement

At recursion step t, the TRM produces two sets of logits:

- **Reference logits** ℓ_u: from $\mathbf{z}_H^n$, representing the policy before the reasoning step.

- **Conditional logits** ℓ_c: from $\mathbf{z}_H^{n+1}$, after attending to $\mathbf{z}_L^{n+1}$.

Define the **residual** $\Delta\ell(a) = \ell_c(a) - \ell_u(a)$. With guidance weight w, $\pi_w = \text{softmax}(\ell_u + w\Delta\ell)$. Let $y^\star$ denote the correct class at the current decoding position. We ask: under what condition does increasing w improve $\pi_w(y^\star)$?

Proposition 4.1 (Advantage Margin Condition). *Let $\mathcal{L}(w) = -\log\pi_w(y^\star)$ be the cross-entropy loss for the correct class. Then $\frac{d}{dw}\mathcal{L}(w) < 0$ (i.e., the loss strictly decreases as w increases) if and only if*

$$\underbrace{\Delta\ell[y^\star]}_{\textit{Boost to correct class}} \quad > \quad \underbrace{\mathbb{E}_{a\sim\pi_w}\big[\Delta\ell[a]\big]}_{\textit{Policy-weighted average boost}} \quad . \qquad (8)$$

Moreover, $\frac{d}{dw}\mathcal{L}(w) = \mathbb{E}_{a\sim\pi_w}[\Delta\ell[a]] - \Delta\ell[y^\star]$ and $\frac{d^2}{dw^2}\mathcal{L}(w) = \text{Var}_{a\sim\pi_w}[\Delta\ell[a]] \geq 0$.

See Appendix A for details. Intuitively, reasoning helps when the residual gives the correct prediction a larger boost than it gives a typical alternative, in other words, when **Advantage Margin** is positive.

4.2. From Implicit Discovery to Explicit Supervision

In standard TRM training, the model attempts to satisfy Proposition 4.1 implicitly. DIS *explicitly* enforces a positive Advantage Margin at every recursive step by constructing intermediate targets that contract toward the solution.

Let $\mathbf{x}$ be the input and $\mathbf{y}^\star$ the ground-truth output sequence. Let Φ be a target generator (e.g., a discrete diffusion-style schedule) that produces a sequence $\{\mathbf{y}_s^\dagger\}_{s=0}^{N_{\text{sup}}}$ with $\mathbf{y}_{N_{\text{sup}}}^\dagger = \mathbf{y}^\star$ and each $\mathbf{y}_s^\dagger$ strictly closer to $\mathbf{y}^\star$ than $\mathbf{y}_{s-1}^\dagger$ under a fixed metric.

Ideally, we want the *reference* logits at step s to match the previous target $\mathbf{y}_{s-1}^\dagger$ and the *conditional* logits to match the improved target $\mathbf{y}_s^\dagger$, leading to a double-loss objective:

$$\mathcal{L}_{\text{Dual}} = \sum_{s=1}^{N_{\text{sup}}} \Big[\underbrace{\text{CE}(\ell_u^s, \mathbf{y}_{s-1}^\dagger)}_{\textit{Anchor previous}} + \underbrace{\text{CE}(\ell_c^s, \mathbf{y}_s^\dagger)}_{\textit{Predict current}} \Big]. \qquad (9)$$

However, in a recursive architecture, this is redundant: the input to step s (which produces ℓ_u^s) is a deterministic function of the output of step $s-1$ (which produced ℓ_c^{s-1}). Since step $s-1$ is trained toward $\mathbf{y}_{s-1}^\dagger$, ℓ_u^s is implicitly anchored. We therefore use the *single-loss* objective

$$\mathcal{L}_{\text{DIS}} = \sum_{s=1}^{N_{\text{sup}}} \text{CE}(\ell_c^s, \mathbf{y}_s^\dagger). \qquad (10)$$

Because optimization is sequential, minimizing Eq. (10) is (under teacher-forced recursion) inductively equivalent to minimizing Eq. (9): forcing $\ell_c^{s-1} \to \mathbf{y}_{s-1}^\dagger$ ensures ℓ_u^s starts near $\mathbf{y}_{s-1}^\dagger$.

Training ℓ_c on $\mathbf{y}_s^\dagger$ while ℓ_u is anchored to $\mathbf{y}_{s-1}^\dagger$ forces the residual $\Delta\ell = \ell_c - \ell_u$ to encode the transition from $\mathbf{y}_{s-1}^\dagger$ to $\mathbf{y}_s^\dagger$. This yields our second formal result.

Proposition 4.2 (Guaranteed Improvement). *Assume the target generator Φ provides strictly improving targets with respect to a fixed scoring distribution P, i.e., $\log\frac{P(\mathbf{y}_s^\dagger)}{P(\mathbf{y}_{s-1}^\dagger)} > 0$ for all s. Then minimizing the sequential loss $\mathcal{L}_{DIS}$ drives the expected Advantage Margin to be positive:*

$$\mathbb{E}\Big[\Delta\ell[y^\star] - \mathbb{E}_{a\sim\pi_w}\Delta\ell[a]\Big] \ > \ 0,$$

where the outer expectation is over data (and model stochasticity) and the inner expectation is with respect to π_w for any fixed $w \geq 0$.

See Appendix A for details. Standard training *hopes* the model learns to improve; DIS *forces* each residual vector $\Delta\ell$ to point toward the solution. Consequently, DIS turns TRM from a black-box recurrent model into a verifiable iterative refinement procedure in which every forward pass is supervised to act as a precise policy-improvement step.

4.3. Algorithm

In standard TRM training, the model attempts to satisfy Theorem 4.1 implicitly. It tries to find a latent configuration $\mathbf{z}_L$ such that the final margin is positive. This is often unstable because the gradient signal must traverse the entire recursive chain.

DIS solves this by *explicitly* enforcing a positive Advantage Margin at every recursive step. We achieve this by constructing a sequence of intermediate targets that strictly contract towards the solution. Let $\{\mathbf{y}_s^\dagger\}_{s=1}^{N_{\text{sup}}}$—constructed so that the expected discrepancy to $\mathbf{y}^\star$ strictly decreases with s. Let $\mathbf{x}\in\mathcal{X}$ be the input, $\mathbf{y}^\star\in\mathcal{Y}$ the final target, and $(\mathbf{y},\mathbf{z})$ the TRM state passed across supervision steps. A *target generator* Φ produces a sequence of stepwise targets

$$\mathbf{y}_1^\dagger, \ \mathbf{y}_2^\dagger, \ \ldots, \ \mathbf{y}_{N_{\text{sup}}}^\dagger \ = \ \Phi(\mathbf{x}, \mathbf{y}^\star; \ 1{:}N_{\text{sup}}), \qquad (11)$$

with $\mathbf{y}_{N_{\text{sup}}}^\dagger = \mathbf{y}^\star$, such that the distance to the final solution decreases monotonically along the schedule, e.g. where d is a task-appropriate discrepancy (e.g., Hamming distance over tokens). Each supervision step $s \in \{1, \ldots, N_{\text{sup}}\}$ is indexed and receives a *puzzle embedding* $E(p)$. Our algorithm admits diverse sources of intermediate targets Φ:

1. **Programmable edits.** Deterministic code-based generator creates path from input to output, for example puzzle solvers that reveal N constraint-consistent move per step, yielding $\mathbf{y}_{s+1}^\dagger = \text{Edit}(\mathbf{y}_s^\dagger)$.

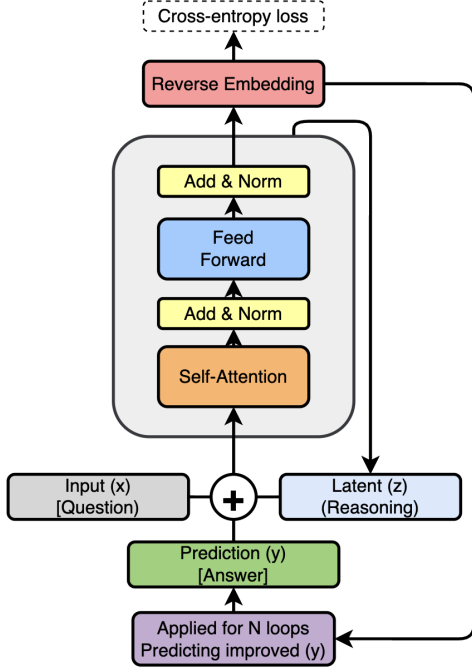2. **LLM-generated plans.** A teacher LLM proposes a sequence of intermediate solutions or sketches; these

Figure 2. DIS model architecture. Algorithm starts with the embedded input question $\mathbf{x}$, initial embedded answer $\mathbf{y}$, and latent state z. For up to n improvement steps, it tries to improve its answer $\mathbf{y}$ by simulating a discrete diffusion process, addressing any errors from its previous answer in an parameter-efficient manner.

```
1  def latent_reasoning(x, y, z, n=2):
2      with torch.no_grad():
3          for j in range(T-1):
4              for i in range(n):
5                  z = net(x, y, z)
6              y = net(y, z)
7      for i in range(n):
8          z = net(x, y, z)
9      y = net(y, z)
10     return (y.detach(), z.detach()),
           output_head(y)
11
12 # Deep Improvement Supervision
13 for x_input, y_true in train_dataloader:
14     y, z = y.init, z.init
15     for step in range(N_supervision):
16         y_step = f(x_true, y_true, step)
17         x = input_embedding(x_input, step)
18         (y, z), y_hat = latent_reasoning(x
               , y, z)
19         loss = softmax_cross_entropy(y_hat
               , y_step)
20         loss.backward()
21         opt.step()
22         opt.zero_grad()
```

Figure 3. Pseudocode for reasoning with deep improvement supervision. With $T = 1$ (as in our *medium* settings), **we avoid the large (`no-grad`) cycle** and significantly reduce computational time.

are projected onto the task's discrete output space to form $\{\mathbf{y}_s^\dagger\}$.

3. **Discrete diffusion schedules.** Define a corruption process $q_\beta(\tilde{\mathbf{y}} \mid \mathbf{y}^\star)$ (e.g., token masking or random replacement with rate β). Choose a decreasing noise schedule and sample $\mathbf{y}_s^\dagger \sim q_{\beta_s}(\cdot \mid \mathbf{y}^\star)$ so that the targets become progressively less corrupted, approximating a reverse-diffusion path over discrete outputs[2].

These constructions guarantee by design (or in expectation) that the improvement direction is explicit. In our paper, we choose the simplest version with stepwise targets via discrete corruption. Let $\mathbf{x}$ be the input and $\mathbf{y}^\star$ the ground-truth output. Choose a decreasing noise schedule $0 = \beta_{N_{\text{sup}}} < \cdots < \beta_2 < \beta_1 \leq 1$ and a token-level corruption kernel q_β (e.g., masking at rate β). Define a sequence of intermediate targets

$$\mathbf{y}_s^\dagger \sim q_{\beta_s}(\cdot \mid \mathbf{y}^\star), \qquad s = 1, \ldots, N_{\text{sup}}, \qquad (12)$$

so that $\mathbf{y}_{N_{\text{sup}}}^\dagger = \mathbf{y}^\star$ and, in expectation, the discrepancy with $\mathbf{y}^\star$ (e.g., the Hamming distance) decreases with s. We pass $(\mathbf{y}_{s+1}, \mathbf{z}_{s+1})$ to the next deep supervision step:

---

[2]We do not position our method as diffusion models. It does not employ diffusion-based schedulers, nor does it explicitly compute concrete scores or distributional ratios (Lou et al., 2023). We propose a general framework for guiding the reasoning steps.

During training, the model receives step-by-step supervision targets and computes losses at each improvement step, allowing it to learn how to sequentially enhance its answers through backpropagation across the reasoning chain.

In this framework, a diffusion corruption process is merely one option for supervising the reasoning process. As demonstrated above, any other sampler for improvement guidance may be used. Crucially, we incorporate the time step t into the model input and observe that an integer-based time index $(0, 1 \ldots, N_{sup})$ yields superior results compared to standard continuous diffusion time conditioning $t \in [0, 1]$. Simplification vs. TRM are:

- **Recursion budget:** DIS: $T{=}1, n{=}2$ vs. TRM: $T{=}3, n{=}6$ on ARC—DIS backpropagates through one cycle with **two** internal latent/answer updates; TRM runs no-grad cycles before a grad cycle. The total step formula is $N_{sup}[T(n + 1)]$, so TRM does $16 * [3 * (6+1)] = 336$, while we have $6(2+1) = \mathbf{18}$

- **Supervision:** DIS trains each step toward a step-specific target $\mathbf{y}_s^\dagger$ that provably induces monotone improvement; TRM supervises every step directly on the final label $\mathbf{y}^\star$.

- **Halting/ACT:** DIS uses a **fixed** $N_{\text{sup}}{=}6$ with **no halting head** and no extra forward pass; TRM/HRM use

halting (HRM's ACT requires a second forward pass for the continue loss).

- **Backbone & pipeline:** We keep TRM's attention backbone and augmentation/evaluation pipeline for a fair comparison on `ARC`, as self-attention generalizes better on 30×30 puzzles.

Our DIS experiments adopt a minimal-compute design: we retain TRM's tiny 2-layer attention backbone and ARC protocol but use only **six** supervised improvement steps with a **single external cycle** (comprising two internal updates) and **no halting head**. This streamlined setup isolates the contribution of guided, monotonic improvement.

5. Experiments

In this section, we provide a detailed explanation and results on the complex `N-Queens`, `Sudoku-Extreme` and `ARC-AGI` problems.

Backbone. Our DIS model reuses the *tiny single-network* TRM backbone but eliminates TRM's extra recursion and halting heads. We use a 2-layer Transformer block with RMSNorm, SwiGLU MLPs, and rotary position embeddings; weights are shared for both the latent-update and answer-update calls, exactly as in TRM's attention variant ("TRM-Att"), to isolate the contribution of DIS from capacity and architecture differences (Jolicoeur-Martineau, 2025). The task specific hyperparameters as the hidden layers size and reasoning steps are presented below, per task protocol.
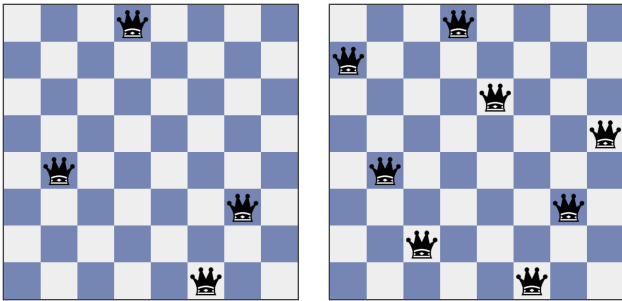
5.1. `N-Queens`

Figure 4. N-Queens reasoning problem example. Left is input and right is target solution.

Task Format. The `N-Queens` problem is a combinatorial reasoning task that involves placing Q queens on a 8×8 chessboard (Oarga & Du, 2025). The fundamental objective is to arrange the queens such that no two queens threaten each other, which imposes the strict constraint that no two queens can share the same row, column, or diagonal. This problem serves as a benchmark for evaluating a model's ability to generate valid configurations under complex constraints.

The complexity of the task is directly determined by the parameter Q, which dictates the total number of queens that must be accommodated on the board. Complexity levels corresponding to problem instances ranging from $Q = 1$ to $Q = 7$ queens that are already on board, with lower values of Q representing increasingly difficult reasoning challenges. In our experiments, we randomly sampled train and test experience with different complexity.

Only the classic 8×8 board and no augmentation was used, an example of input and target is represented in 4. We generated 7,200 train and 200 test examples with a sequence length of 64 and a vocabulary size of 3.
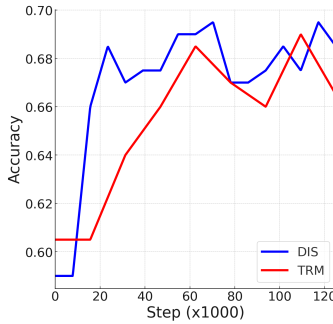

Figure 5. Accuracy curves on `N-Queens` problem.

Results. The purpose of this experiment was to assess the impact of DIS training. As mentioned in §4, we used T = 1 and n = 2 for DIS, while the regular TRM of T = 3 and n = 6 was applied, with a halting head and 16 supervision steps. We achieved the same 0.69 accuracy while using much fewer inference steps. The architectures of the 0.8 mill parameters were used for both methods.

5.2. `ARC` Evaluation Protocol

Task format. `ARC` puzzles are sets of colored grids with 2–3 input–output demonstrations and 1–2 test inputs per task; the maximum grid size is 30×30. Accuracy is scored over all test grids with two attempts permitted per task (standard ARC scoring). We evaluate on the public evaluation sets of `ARC-AGI-1` (800 tasks) and `ARC-AGI-2` (1,120 tasks), following TRM.

Data augmentation. We adopt the augmentation pipeline of TRM to mitigate small-data overfitting: 1000 augmentations per puzzle via color permutations, dihedral-group transforms (90° rotations, flips/reflections) and translations. As in TRM, we also include the 160 ConceptARC tasks as additional training puzzles (Moskvichev et al., 2023). We attach a puzzle-specific embedding token per instance.

Pre-/post-processing. Inputs and outputs are tokenized as discrete color IDs; we concatenate demonstrations and the target input in the same sequence layout used by TRM, our positional scheme match theirs. For evaluation purposes, we apply the majority vote of the TRM to 1,000 augmented inferences per puzzle. Different number of passes are used and we report pass@2 as it is the most common setting.

5.3. ARC Model Settings

Objective. Each supervision step $s \in \{1, \dots, 6\}$ is trained toward a *step-specific intermediate target* $\mathbf{y}_s^\dagger$ produced by a discrete corruption schedule of ground truth $\mathbf{y}^\star$ with monotonically decreasing noise. We use token-masking/replacement with a linearly decreasing mask rate over the 6 steps so that $\mathbb{E}[d(\mathbf{y}_s^\dagger, \mathbf{y}^\star)]$ decreases with s. The loss is standard token-level cross-entropy on $f_O(\mathbf{y})$ against $\mathbf{y}_s^\dagger$, with linearly increasing step weights w_s to emphasize late-step fidelity.

Optimization. We follow TRM's stable training recipe wherever applicable: Adam-Atan with $\beta_1{=}0.9$, $\beta_2{=}0.95$, a 2k-step warm-up, and the stable-max cross-entropy variant for stability. For ARC experiments, we use weight decay 0.1 and we did not find EMA important.

We match the hidden size of the TRM $D{=}512$ **and call it the medium model** in Table 1. When we use $D{=}256$ and the single decoder layer model, which results in 0.8 mil. parameters, **we call it compact**. Also, we match batch sizing; embedding LR warm-up and an elevated embedding LR (as in TRM) are retained.

Deep improvement supervision loop. For each mini-batch we run $N_{\text{sup}}{=}6$ DIS steps. At each step, we execute a single external cycle (since $T{=}1$) comprising two internal latent/answer updates ($n{=}2$), backpropagating through the full cycle; we then detach $(\mathbf{y}, \mathbf{z})$ before the next step. We **do not** train a halting/ACT head.

Importantly, when using a discrete diffusion model, the supervision trajectories are generated/sampled on the fly, as in a regular diffusion process. Therefore, for the same task, *we can have various diffusion steps* towards the target.

Test-time compute. We run the **same $N_{\text{sup}}{=}6$ steps** at evaluation. To compare fairly with prior ARC protocols, we keep TRM's test-time augmentation vote: run the model across 1000 geometric/color augmentations of a puzzle and return the most common prediction.

5.4. ARC Results

For our experiments, we replicated the TRM experiments and achieved slightly lower results than those reported in the original paper. We also re-implemented TRM with the same hyperparameter settings as in our *medium* model to compare the methods with identical resources, we set $T = 1$, $n = 2$, but still use $N_{\text{sup}}{=}16$ for TRM, because the halting mechanism remained active. In addition, we implemented a smaller network to reproduce a compact model consisting of only 0.8 million parameters.

The results are presented in Table 1 and Figure 6. As shown, for the compact model we dramatically outperform the original TRM. This shows that for TRM, latent reasoning steps

Table 1. Model Performance Comparison, pass@2

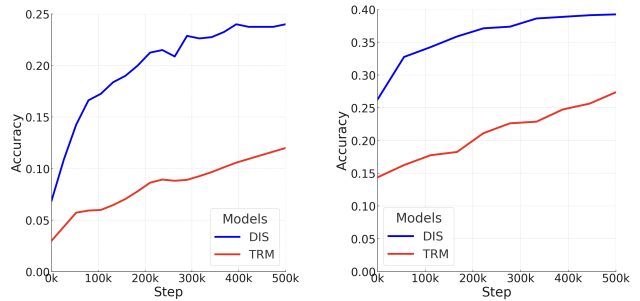| Method	Params	ARC-1	ARC-2
Chain-of-thought, pretrained			
Deepseek R1	671B	15.8	1.3
Claude 3.7 16K	?	28.6	0.7
o3-mini-high	?	34.5	3.0
Gemini 2.5 Pro 32K	?	37.0	4.9
Grok-4-thinking	1.7T	66.7	16.0
Bespoke (Grok-4)	1.7T	**79.6**	**29.4**
Small-sample training			
TRM-compact	0.8M	12.0	0.0
DIS-compact (Ours)	0.8M	24.0	0.0
TRM-medium	7M	27.1	0.0
DIS-medium (Ours)	7M	40.0	3.0
TRM	7M	40.4	3.0
DIS	7M	40.0	3.0

Figure 6. The DIS and TRM models pass@2 scores under the compact (left) and medium (right) setups.

are important. We reduced the total number of latent steps nine times and achieved a significant improvement in performance. However, explicit supervision of each step can overcome this drawback by simplifying the task for the TRM, meaning that longer latent reasoning is unnecessary. Furthermore, our medium model outperforms the medium TRM and achieves results comparable to the original TRM.

Shaped credit assignment across supervision steps. In baseline TRM, every step is trained directly against $\mathbf{y}^\star$, leaving it to the model to discover a self-improvement curriculum DIS supplies *explicit* intermediate targets $\{\mathbf{y}_s^\dagger\}$, aligning the step-s gradients with a concrete improvement objective. This reduces the burden on the latent state $\mathbf{z}$ to implicitly encode a stepwise plan and can accelerate optimization in scarce-data regimes, where TRM has been shown to be the most effective. DIS retains TRM's minimal two-feature interface $(\mathbf{y}, \mathbf{z})$, single tiny network reused for both updates, and the schedule of $T{-}1$ no-grad cycles followed by one grad cycle. It inherits the simplicity advantages of TRM while changing only the supervision signal.

Compute and stability. With a monotone schedule, DIS turns each supervision step into a measurable sub-goal. We preserves TRM's compute profile per step (one gradient-bearing recursion cycle) and and avoid HRM/TRM-style ACT. If targets are generated offline, the runtime overhead is negligible; if produced online (e.g., by a teacher model), they can be cached or amortized across epochs. For training we used the same 4 `GPU H100` setting as TRM, but learning takes ≈ 40 hours against 72 in TRM.

6. Discussion

Potential algorithmic improvements. A key limitation of the current implementation is the use of a fixed number of supervision steps for every `ARC` task. However, task complexity varies significantly; some tasks may benefit from a higher number of denoising steps, while others require fewer. This observation aligns with findings from the original TRM paper, which highlighted the significant contribution of the halting mechanism to final performance. Therefore, explicitly predicting the necessary number of denoising steps for each task could potentially enhance overall model efficiency and accuracy.

Another promising direction for technical improvement involves adopting a discrete latent space. This approach has been successfully utilized in deep learning architectures such as the Dreamer model (Hafner et al., 2019) and VQ-VAE (Razavi et al., 2019), where latent spaces have proven to be robust and scalable for generative tasks.

Alternative Improvement Generators. As outlined in §4.3, there are several viable methods for generating intermediate steps. While the discrete diffusion prior (Figure 7) serves as the primary source in this work, our framework is designed to support various step-generation approaches.

We also investigated the use of LLM-generated trajectories between transition samples and their targets, specifically utilizing the Gemini 2.5 Pro model. We trained a compact network (0.8 million parameters) on these trajectories; however, this method underperformed compared to the diffusion prior.

We hypothesize that LLM-generated trajectories fail to provide a monotonic improvement path, often introducing highly nonlinear "jumps" between intermediate steps that are difficult for a small model to capture.

Consequently, the model trained with LLM improvement supervision achieved only 10% accuracy, compared to the 24% achieved with the diffusion prior. Exploring code-based generation of intermediate steps remains a promising direction for future work to improve the algorithm's performance.

7. Conclusion

We demonstrate that small, iterative reasoning models can achieve competitive performance on complex reasoning tasks such as the Abstraction and Reasoning Corpus, challenging the dominance of large-scale language models. By reinterpreting TRMs through the lens of reinforcement learning, we reveal that TRMs implicitly perform policy improvement, where a latent "working memory" state guides the model toward better solutions over recursive steps.

The key contribution is Deep Improvement Supervision, builds on this insight by introducing a structured, stepwise training regime. DIS provides intermediate targets through a discrete diffusion process, transforming the challenging problem of long-term credit assignment into a more tractable supervised learning task (Ho & Salimans, 2022; Frans et al., 2025). This approach not only simplifies training by eliminating the need for learned halting mechanisms but also enhances efficiency, reducing the number of forward passes by 24x with high accuracy.

References

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

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Frans, K., Park, S., Abbeel, P., and Levine, S. Diffusion guidance is a controllable policy improvement operator. *arXiv preprint arXiv:2505.23458*, 2025.

Giannou, A., Rajput, S., Sohn, J.-y., Lee, K., Lee, J. D., and Papailiopoulos, D. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. PMLR, 2023.

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

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Ho, J. and Salimans, T. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

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

Lou, A., Meng, C., and Ermon, S. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

Moskvichev, A., Odouard, V. V., and Mitchell, M. The conceptarc benchmark: Evaluating understanding and generalization in the arc domain. *Transactions on Machine Learning Research*, 2023. arXiv:2305.07141.

Oarga, A. and Du, Y. Generalizable reasoning through compositional energy minimization. *arXiv preprint arXiv:2510.20607*, 2025.

Razavi, A., Van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

Sutton, R. S., Barto, A. G., et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Wang, G., Li, J., Sun, Y., Chen, X., Liu, C., Wu, Y., Lu, M., Song, S., and Yadkori, Y. A. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yang, L., Lee, K., Nowak, R., and Papailiopoulos, D. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.
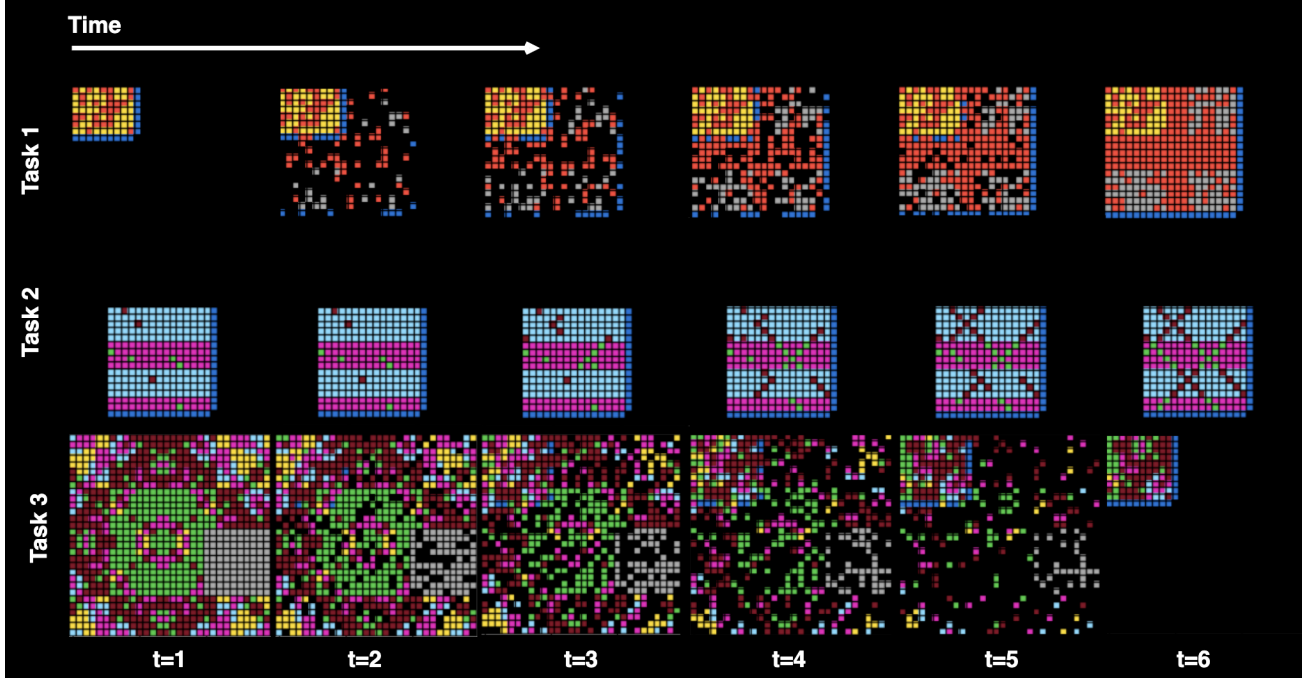
Figure 7. The linear corruption process is shown over six steps, from the initial input at time $t = 0$ to the target at time $t = 6$. A single training sample is illustrated per task.

A. Analysis

Notation and Definitions. Let $\mathbf{x}$ be the input context. At any reasoning step n, the TRM produces two logit vectors over the vocabulary $\mathcal{V}$:

- $\ell_u^n \in \mathbb{R}^{|\mathcal{V}|}$: the *reference* logits (the state *before* the current reasoning step),

- $\ell_c^n \in \mathbb{R}^{|\mathcal{V}|}$: the *conditional* logits (the state *after* the current reasoning step).

We define the **Reasoning Residual** as

$$\Delta\ell^n \;=\; \ell_c^n - \ell_u^n.$$

For a guidance scale $w \geq 0$, define the guided logits $\ell_w^n := \ell_u^n + w\,\Delta\ell^n$ and the corresponding policy

$$\pi_w(a) \;=\; \frac{\exp\big(\ell_u^n[a] + w\,\Delta\ell^n[a]\big)}{Z(w)}, \qquad Z(w) \;=\; \sum_{k \in \mathcal{V}} \exp\big(\ell_u^n[k] + w\,\Delta\ell^n[k]\big).$$

Let $y^\star$ denote the index of the ground-truth correct token at the current position. In the proofs below, we *fix a step n* and drop the superscript n when unambiguous.

We first show the condition under which increasing w strictly reduces the loss.

Proposition A.1 (Advantage Margin Condition). *Let $\mathcal{L}(w) = -\log \pi_w(y^\star)$ be the cross-entropy loss of the correct token $y^\star$. Then $\frac{d}{dw}\mathcal{L}(w) < 0$ if and only if*

$$\Delta\ell[y^\star] \;>\; \mathbb{E}_{a \sim \pi_w}\big[\Delta\ell[a]\big].$$

Moreover,

$$\frac{d}{dw}\mathcal{L}(w) = \mathbb{E}_{a \sim \pi_w}[\Delta\ell[a]] - \Delta\ell[y^\star], \qquad \frac{d^2}{dw^2}\mathcal{L}(w) = \mathrm{Var}_{a \sim \pi_w}\big[\Delta\ell[a]\big] \geq 0.$$

Proof. Write $\ell_w = \ell_u + w\,\Delta\ell$.

$$\mathcal{L}(w) = -\log\left(\frac{\exp(\ell_w[y^\star])}{Z(w)}\right) = -\ell_w[y^\star] + \log Z(w)$$

$$= -\left(\ell_u[y^\star] + w\,\Delta\ell[y^\star]\right) + \log\sum_{k\in\mathcal{V}}\exp\left(\ell_u[k] + w\,\Delta\ell[k]\right).$$

The derivative of the linear term is $-\Delta\ell[y^\star]$. Let $E_k = \exp(\ell_u[k] + w\,\Delta\ell[k])$, so $Z(w) = \sum_k E_k$. Then

$$\frac{d}{dw}\log Z(w) = \frac{1}{Z(w)}\sum_k E_k\,\Delta\ell[k] = \sum_k \frac{E_k}{Z(w)}\,\Delta\ell[k] = \mathbb{E}_{a\sim\pi_w}\big[\Delta\ell[a]\big].$$

Combining the two derivatives yields

$$\frac{d}{dw}\mathcal{L}(w) = -\Delta\ell[y^\star] + \mathbb{E}_{a\sim\pi_w}\big[\Delta\ell[a]\big],$$

which is negative exactly when the stated inequality holds. Differentiating once gives $\frac{d^2}{dw^2}\mathcal{L}(w) = \mathrm{Var}_{a\sim\pi_w}[\Delta\ell[a]] \geq 0$. $\square$

Thus, reasoning (the residual $\Delta\ell$) helps iff it boosts the correct class $y^\star$ *more than* it boosts a typical alternative (the policy-weighted average).

A.1. Global Convergence

While Proposition A.1 guarantees a *local* improvement as w increases, we also analyze the sequential process with a Lyapunov-style argument.

Proposition A.2 (Discrete Lyapunov Contraction). *Let $d(\hat{y}, y^\star)$ be a bounded distance metric (e.g., Hamming distance) between a predicted sequence $\hat{y}$ and the ground truth $y^\star$. Define the potential at step n as $V_n := d(\mathrm{argmax}(\ell_c^n), y^\star)$. Assume:*

1. ***Strictly Improving Targets.** A generator Φ yields $\mathbf{y}_0^\dagger, \ldots, \mathbf{y}_N^\dagger$ with*

$$d(\mathbf{y}_n^\dagger, y^\star) \leq d(\mathbf{y}_{n-1}^\dagger, y^\star) - \delta \quad \text{for some } \delta > 0.$$

2. ***Training Minimization.** The model minimizes $\mathcal{L}_n = \mathrm{CE}(\ell_c^n, \mathbf{y}_n^\dagger)$ so that $\mathrm{argmax}(\ell_c^n) = \mathbf{y}_n^\dagger$ with high probability.*

Then $V_n - V_{n-1} \leq -\delta < 0$. Consequently, V_n decreases monotonically and hits 0 in at most $\lceil V_0/\delta\rceil$ steps.

Proof. By Assumption 2, $\mathrm{argmax}(\ell_c^n) \approx \mathbf{y}_n^\dagger$, hence $V_n \approx d(\mathbf{y}_n^\dagger, y^\star)$ and $V_{n-1} \approx d(\mathbf{y}_{n-1}^\dagger, y^\star)$. Assumption 1 gives $d(\mathbf{y}_n^\dagger, y^\star) \leq d(\mathbf{y}_{n-1}^\dagger, y^\star) - \delta$, therefore $V_n \leq V_{n-1} - \delta$. Since $V_n \geq 0$, the bound on the hitting time follows. $\square$

Proposition A.3 (Guaranteed Improvement). *Assume the target generator Φ provides strictly improving targets such that the likelihood ratio $\log\frac{P(\mathbf{y}_n^\dagger)}{P(\mathbf{y}_{n-1}^\dagger)} > 0$ (with respect to a fixed scoring distribution P over sequences). Then minimizing the sequential loss $\mathcal{L}_{DIS}$ drives the* expected *Advantage Margin to be positive:*

$$\mathbb{E}\big[\,\Delta\ell[y^\star] - \mathbb{E}_{a\sim\pi_w}\Delta\ell[a]\,\big] > 0, \qquad \text{for any fixed } w \geq 0.$$

Proof. **1. Optimal-logit approximation.** Assume sufficient capacity and that $\mathcal{L}_{\mathrm{DIS}}$ is minimized. Then for some constants C_1, C_2 (independent of a),

$$\ell_c^n \approx \log P(\cdot \mid \mathbf{y}_n^\dagger) + C_1, \qquad \ell_u^n \approx \log P(\cdot \mid \mathbf{y}_{n-1}^\dagger) + C_2.$$

2. Residual as a likelihood ratio. Taking the difference eliminates constants:

$$\Delta\ell = \ell_c^n - \ell_u^n \approx \log\frac{P(\cdot \mid \mathbf{y}_n^\dagger)}{P(\cdot \mid \mathbf{y}_{n-1}^\dagger)}.$$

3. Margin at the ground-truth class. For the correct token $y^\star$,

$$\Delta\ell[y^\star] \;\approx\; \log \frac{P(y^\star \mid \mathbf{y}_n^\dagger)}{P(y^\star \mid \mathbf{y}_{n-1}^\dagger)} \;>\; 0$$

by the "strictly improving targets" premise.

4. Policy-weighted average. As the targets sharpen toward $y^\star$, probability mass shifts into $y^\star$ and away from incorrect tokens. Consequently, under any fixed π_w (with $w \geq 0$), $\mathbb{E}_{a \sim \pi_w} \Delta\ell[a] < \Delta\ell[y^\star]$. Taking expectation over data (and model stochasticity) yields the stated inequality. $\qquad\square$