Deep Improvement Supervision

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

Abstract

Recent work has demonstrated 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 asymmetric latent reasoning of TRMs as both an implicit policy improvement algorithm and a form of classifier-free diffusion guidance. 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.

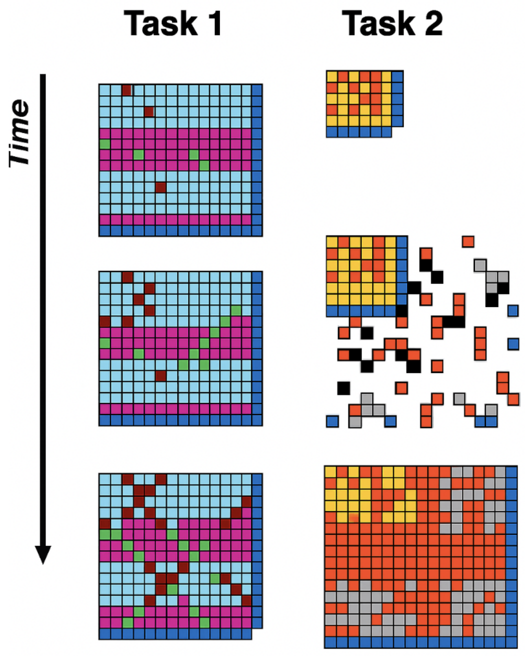

Figure 1. Blueprint of the discrete diffusion process on the `ARC`. Starting from the input x, following timestep t, we generate diffusion steps to the target y (Chollet, 2019).

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 reasoning 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

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

Preprint. November 20, 2025.

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?*

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 modules a high-frequency "low-level" module f_L and a low-frequency "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:

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

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 $\mathbf{z}_H$: $\hat{\mathbf{y}} = \arg\max f_O(z_H)$). $\mathbf{z}_L$ is a "working memory" state that's updated using the input 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 network**, each recursion:

$$\begin{aligned} \mathbf{z}_L^{n+1} &\leftarrow f_L^\phi((\mathbf{z}_L^n + \mathbf{x}) + \mathbf{z}_H), \quad \text{(repeated } n \text{ times)} \\ \mathbf{z}_H &\leftarrow f_H^\phi(\mathbf{z}_L^{n+1} + \mathbf{z}_H), \\ \hat{\mathbf{y}} &= \arg\max f_O(\mathbf{z}_H). \end{aligned} \tag{2}$$

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 $\mathbf{z}_L$ looks like non-sensical tokens.

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 $(x, \mathbf{y}, \mathbf{z})$; 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 and removes the ACT *continue* head.

TRM also uses an exponential moving average (EMA) of weights for stability on small data. Let $\mathcal{L}_{\text{task}} = \mathrm{CE}(f_O(\mathbf{y}), \mathbf{y}_{\text{true}})$ be the prediction loss and $\mathcal{L}_{\text{halt}} = \mathrm{BCE}(q(\mathbf{y}), \hat{\mathbf{y}} = \mathbf{y}_{\text{true}})$ 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

At the heart of RL is the idea of optimizing beyond the performance shown in the data, while RL methods are unstable in training. 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 analysis of diffusion models as RL methods. The core construction, termed classifier-free guidance RL (CFGRL), parameterizes a target policy as:

$$\pi(\mathbf{y} \mid \mathbf{s}) \;\propto\; \hat{\pi}(\mathbf{y} \mid \mathbf{s})\; f\big(A_{\hat{\pi}}(s, \mathbf{y})\big), \tag{3}$$

where $f : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a nonnegative monotonically increasing function of the advantage $A^{\pi}(\mathbf{s}, \mathbf{y}) = Q^{\pi}(\mathbf{s}, \mathbf{y}) - V^{\pi}(\mathbf{s})$, 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 (3) 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. (3) can now be equivalently defined as: $\pi(\mathbf{y} \mid \mathbf{s}) \;\propto\; \hat{\pi}(\mathbf{y} \mid \mathbf{s}) p(\mathbf{o}|\mathbf{s}, \mathbf{y})$.

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

$$\nabla_a \log \pi_w(\mathbf{y} \mid \mathbf{s}) \;=\; \nabla_a \log \hat{\pi}(\mathbf{y} \mid \mathbf{s}) + w\Big(\nabla_a \log \hat{\pi}(\mathbf{y} \mid \mathbf{s}, \mathbf{o}{=}1) - \nabla_a \log \hat{\pi}(\mathbf{y} \mid \mathbf{s})\Big), \tag{4}$$

which corresponds to sampling from $\pi_w \propto \hat{\pi}(\mathbf{y} \mid \mathbf{s})\, p(\mathbf{o}{=}1 \mid s, \mathbf{y})^w$. Please see (cite) for derivation other 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 Tiny Recursive Model (TRM) to demonstrate its relationship with well-established frameworks in generative modeling and reinforcement learning. The goal of this section is to provide a formal hypothesis on *why* TRM works. We aim to build a bridge between the empirical success of recursive reasoning and existing mathematical theory, specifically connecting the TRM's asymmetric structure to **Classifier-Free Guidance (CFG)** and **Implicit Policy Improvement**.

Notation. Let $s = (x, \mathbf{z}_L, \mathbf{z}_H)$ be the state, where x is the input, $\mathbf{z}_L$ is the latent reasoning *scratchpad*, and $\mathbf{z}_H$ is the current embedded solution. Despite the fact that TRM considering two modules f_L^{ϕ} and f_H^{ϕ}, the same netwroks without any changes are using for both, so we simply write f_{ϕ}. An output head is f_O. As established in §2.2, only $\mathbf{z}_H$ is decoded and penalized. We introduce an optimality variable $\mathbf{o} \in \{\emptyset, 1\}$ to distinguish between two distinct modes of operation:

- **Unconditional / Reference ($\mathbf{o} = \emptyset$):** The policy derived solely from the current solution state, without the benefit of the latest reasoning step.
- **Conditional / Optimal ($\mathbf{o} = 1$):** The policy derived after the latent reasoning process has updated the state.

3.1. Latent Reasoning as Classifier-Free Diffusion Guidance

The Hierarchical Reasoning Model (HRM) and the TRM are fundamentally based on iterative latent updates. As discussed in Sections 2.1 and 2.2, this process involves an explicit asymmetry: $\mathbf{z}_L$ (the reasoning) is updated using the input x, and this reasoning conditions the update of $\mathbf{z}_H$ (the solution).

Let us consider a single latent reasoning cycle. At the beginning of the reasoning cycle, we first have the unconditional $\mathbf{z}_H$, then we can define the dynamics as:

$$\mathbf{z}_L^{t+1} = f_\phi(\mathbf{z}_L^t, \mathbf{z}_H, x) \quad \xrightarrow{\text{conditions}} \quad \mathbf{z}_H^+ = f_\phi(\mathbf{z}_H, \mathbf{z}_L^{t+1}). \tag{5}$$

This asymmetry allows us to extract two distinct readouts from the model at any given step. We define these as the Reference Logits (the *old policy* state) and the Optimal Logits (the *new policy* state):

$$\begin{aligned} \textbf{Reference (Old) Logits:} \quad & \ell_\text{u} := f_O(\mathbf{z}_H) \\ \textbf{Conditional (New) Logits:} \quad & \ell_\text{c} := f_O(\mathbf{z}_H^+) \end{aligned}$$

Here, ℓ_u represents the model's best guess *before* applying the reasoning step, while ℓ_c represents the guess *after* incorporating the reasoning from $\mathbf{z}_L$.

By framing the architecture this way, we can interpret the final prediction of the TRM as a form of Classifier-Free Guidance. We define the final logits ℓ_w as a linear interpolation in log-space:

$$\ell_w = \ell_\text{u} + w \cdot \underbrace{(\ell_\text{c} - \ell_\text{u})}_{\text{Reasoning Residual}}, \qquad \pi_w(\cdot \mid \mathbf{s}) = \text{softmax}(\ell_w). \tag{6}$$

In this framework, the term $(\ell_\text{c} - \ell_\text{u})$ represents the **direction of improvement** identified by the reasoning process. The scalar w acts as a guidance scale (or reasoning depth).

Practical Note: *In standard TRM inference, we typically do not explicitly tune w; instead, the model decodes directly from $\mathbf{z}_H^+$ (equivalent to ℓ_c). However, this can be viewed as a special case where the guidance is implicitly fixed. Equation (6) exposes w as a controllable knob, allowing us to amplify or attenuate the reasoning signal at test time.*

3.2. Latent Reasoning as Implicit Policy Improvement

In Section 3.1, we established that TRM behaves mechanically like Classifier-Free Guidance. We now demonstrate that this mechanism is a practical implementation of a controllable policy improvement operator. The core idea is to generate a new, better policy by combining a reference policy with a factor that encodes "optimality." We frame this as sampling from a "Product Policy":

$$\pi_\text{new}(\mathbf{y}|\mathbf{s}) \;\propto\; \underbrace{\hat{\pi}(\mathbf{y}|\mathbf{s})}_{\text{Reference Policy}} \cdot \underbrace{p(\mathbf{o} = 1|\mathbf{s}, \mathbf{y})}_{\text{Optimality Likelihood}}. \tag{7}$$

Here, $p(\mathbf{o} = 1|\mathbf{s}, \mathbf{y})$ represents the probability that a given action leads to an optimal outcome.

Why TRM-CFG does Policy Improvement. A key result from recent work in guidance-based RL is that product policies guarantee improvement under mild conditions. Specifically, if the optimality likelihood $p(\mathbf{o} = 1|\mathbf{s}, \mathbf{y})$ is monotonically increasing with respect to the Advantage function $A_{\hat{\pi}}(\mathbf{s}, \mathbf{y})$ (i.e., actions that are "better than average" have a higher probability of being optimal), then sampling from the product policy yields a higher expected return than the reference policy.

Crucially, we do not need to learn $p(\mathbf{o} = 1|\mathbf{s}, \mathbf{y})$ explicitly. Instead, we can utilize Bayes' rule to invert the optimality distribution into a conditional policy, as derived in the CF-GRL framework. We can express the log-probability of the improved policy as:

$$\log \pi_\text{new}(\mathbf{y}|\mathbf{s}) \;\propto\; \log \hat{\pi}(\mathbf{y}|\mathbf{s}) \;+\; w \cdot \log p(\mathbf{o} = 1|\mathbf{s}, \mathbf{y})$$

Applying Bayes' rule: $\log p(\mathbf{o}|\mathbf{s}, \mathbf{y}) = \log \hat{\pi}(\mathbf{y}|\mathbf{s}, \mathbf{o}) - \log \hat{\pi}(\mathbf{y}|\mathbf{s}) + C$, we get:

$$\begin{aligned} \log \pi_\text{new}(\mathbf{y}|\mathbf{s}) \;\propto\; & \underbrace{\log \hat{\pi}(\mathbf{y}|\mathbf{s})}_{\text{Old Policy } (\ell_u)} \;+ \\ & w \cdot (\underbrace{\log \hat{\pi}(\mathbf{y}|\mathbf{s}, \mathbf{o} = 1)}_{\text{New Policy } (\ell_c)} - \underbrace{\log \hat{\pi}(\mathbf{y}|\mathbf{s})}_{\text{Old Policy } (\ell_u)}). \end{aligned} \tag{8}$$

This derivation reveals a precise identity between TRM and policy improvement:

$$\boxed{\pi_w(\mathbf{y}|\mathbf{s}) \;\propto\; \hat{\pi}(\mathbf{y}|\mathbf{s}) \cdot \left(\frac{\hat{\pi}(\mathbf{y}|\mathbf{s}, \mathbf{o} = 1)}{\hat{\pi}(\mathbf{y}|\mathbf{s})} \right)^w} \tag{9}$$

By training the model's "Conditional Path" ($\mathbf{z}_H^+$) to predict the correct answer while the "Reference Path" ($\mathbf{z}_H$) predicts the baseline, the residouble term $(\ell_c - \ell_u)$ functionally plays the same role as the advantage-based update in CFGRL. It effectively re-weights the distribution to favor actions that the reasoning step identified as "better," allowing the model to perform a step of policy iteration at every forward pass without explicitly estimating a value function.

4. Deep Improvement Supervision

While Section 3.2 establishes that TRMs implicitly perform policy improvement, relying solely on terminal supervision creates a difficult optimization landscape. In the standard setting, the model must *implicitly* discover a sequence of latent updates that result in a correct final answer. This requires the model to infer the direction of improvement (the advantage) from a sparse, high-variance signal at the end of a long reasoning chain. To solve this issue, we introduce **Deep Improvement Supervision (DIS)**. We first derive the precise mathematical condition required for TRM reasoning to improve performance: *Advantage Margin*. We then demonstrate how DIS directly optimizes this condition, transforming the abstract goal of reasoning into a concrete, step-wise supervised objective.

4.1. The Mechanics of Improvement

To motivate our approach, we must first formalize exactly *how* the asymmetric readout of a TRM leads to better predictions. Recall that at any step s, the TRM produces two sets of logits:

- **Reference Logits (ℓ_u):** Derived from the current state $\mathbf{z}_H$, representing the policy before the reasoning step.
- **Conditional Logits (ℓ_c):** Derived from the updated state $\mathbf{z}_H^+$, which has attended to the latent $\mathbf{z}_L$.

We define the **residual** $\Delta\ell$ as the difference between these views: $\Delta\ell(a) = \ell_c(a) - \ell_u(a)$. When we apply reasoning depth (or guidance) with weight w, the final policy becomes $\pi_w = \text{softmax}(\ell_u + w\Delta\ell)$. We ask, under what conditions does increasing reasoning depth w strictly improve the probability of the correct answer $\mathbf{y}$?

Proposition 4.1 (The Advantage Margin Condition)**.** *Let $\mathcal{L}(w) = -\log \pi_w(\mathbf{y})$ be the cross-entropy loss for the correct class $\mathbf{y}$. The loss strictly decreases as reasoning depth w increases if and only if:*

$$\underbrace{\Delta\ell[\mathbf{y}]}_{\textit{Boost to Correct Class}} > \underbrace{\mathbb{E}_{a\sim\pi_w}[\Delta\ell[a]]}_{\textit{Average Boost}}. \tag{10}$$

Proof. The gradient of the loss with respect to w is given by $\frac{d}{dw}\mathcal{L}(w) = \mathbb{E}_{a\sim\pi_w}[\Delta\ell[a]] - \Delta\ell[\mathbf{y}]$. For the loss to decrease ($\frac{d}{dw}\mathcal{L} < 0$), the residouble of the correct class must exceed the policy-weighted expectation of all residuals. □

This proposition states that for reasoning to be beneficial, the TRM must produce a residouble $\Delta\ell$ that "boosts" the correct answer more than it boosts the average alternative. We call this gap the **Advantage Margin**. If this margin is positive, reasoning deeper is mathematically guaranteed to improve results.

4.2. From Implicit Discovery to Explicit Supervision

In standard TRM training, the model attempts to satisfy Theorem 4.1 implicitly. 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 $\mathbf{x}$ be the input and $\mathbf{y}^\star$ the ground-truth output. Let Φ be a target generator (e.g., a discrete diffusion schedule) that produces a sequence $\{\mathbf{y}_s^\dagger\}_{s=0}^{N_\text{sup}}$ where each target is strictly closer to the ground truth than the last, with $\mathbf{y}_{N_\text{sup}}^\dagger = \mathbf{y}^\star$.

Ideally, we want to enforce that the "Reference" logits match the previous target $\mathbf{y}_{s-1}^\dagger$ and the "Conditional" logits match the improved target $\mathbf{y}_s^\dagger$. This implies a double loss objective:

$$\mathcal{L}_\text{Dual} = \sum_{s=1}^{N_\text{sup}} \Big[\underbrace{\text{CE}(\ell_u^s, \mathbf{y}_{s-1}^\dagger)}_{\text{Anchor Previous}} + \underbrace{\text{CE}(\ell_c^s, \mathbf{y}_s^\dagger)}_{\text{Predict Current}}\Big]. \tag{11}$$

However, this double formulation is computationally redundant in a recursive architecture. In the TRM, the input to step s (which generates ℓ_u^s) is strictly derived from the output of step $s-1$ (which generated ℓ_c^{s-1}). Since step $s-1$ is supervised to minimize the distance to $\mathbf{y}_{s-1}^\dagger$, the term ℓ_u^s is implicitly anchored to the previous target. Therefore, we define our operational Single Loss as:

$$\mathcal{L}_\text{DIS} = \sum_{s=1}^{N_\text{sup}} \text{CE}(\ell_c^s, \mathbf{y}_s^\dagger). \tag{12}$$

Because the optimization is sequential, minimizing Eq. (12) is inductively equivalent to minimizing Eq. (11): by forcing $\ell_c^{s-1} \to \mathbf{y}_{s-1}^\dagger$, we ensure that ℓ_u^s (the starting point of step s) is effectively $\mathbf{y}_{s-1}^\dagger$.

4.3. Why DIS Works

By training ℓ_c on the improved target $\mathbf{y}_s^\dagger$ while ℓ_u is anchored to $\mathbf{y}_{s-1}^\dagger$, we force the residouble $\Delta\ell = \ell_c - \ell_u$ to encode the transition from $\mathbf{y}_{s-1}^\dagger$ to $\mathbf{y}_s^\dagger$. This leads to our second formal result.

Lemma 4.2 (Guaranteed Improvement)**.** *Assume the target generator Φ provides strictly improving targets, such that the likelihood ratio $\log \frac{P(\mathbf{y}_s^\dagger)}{P(\mathbf{y}_{s-1}^\dagger)} > 0$. Minimizing the sequential loss $\mathcal{L}_{DIS}$ drives the expected Advantage Margin to be positive:*

$$\mathbb{E}\big[\Delta\ell[\mathbf{y}^\star] - \mathbb{E}[\Delta\ell]\big] > 0. \tag{13}$$

Standard training hopes the model learns to improve. DIS *forces* the model to improve. By aligning the residouble $\Delta\ell$

with the difference between the current target $\mathbf{y}_s^\dagger$ and the previous target $\mathbf{y}_{s-1}^\dagger$, we mathematically guarantee that the vector $\Delta\ell$ points in the direction of the solution. Consequently, DIS transforms the TRM from a black-box recurrent model into a verifiable iterative refinement algorithm, where each forward pass is supervised to function as a precise policy improvement step.

4.4. 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, as 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

$$\mathbf{y}_1^\dagger,\ \mathbf{y}_2^\dagger,\ \ldots,\ \mathbf{y}_{N_{\text{sup}}}^\dagger \ = \ \Phi(\mathbf{x}, \mathbf{y}^\star;\ 1{:}N_{\text{sup}}), \qquad (14)$$

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 $\mathbf{y}_{s+1}^\dagger = \mathrm{Edit}(\mathbf{y}_s^\dagger)$.
2. **LLM-generated plans.** A teacher LLM proposes a sequence of intermediate solutions or sketches; these 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.

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 $0 = \beta_{N_{\text{sup}}} < \cdots < \beta_2 < \beta_1 \le 1$ and a token-level corruption kernel q_β (e.g., masking at rate β). Define a sequence of intermediate targets

$$\mathbf{y}_s^\dagger \ \sim \ q_{\beta_s}(\cdot \mid \mathbf{y}^\star), \qquad s = 1, \ldots, N_{\text{sup}}, \qquad (15)$$

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

Remark 1: *We do not position our method as diffusion models. Our approach diverges from standard diffusion theory; it does not employ sampling or diffusion-based schedulers, nor does it explicitly compute concrete scores or distributional ratios (Lou et al., 2023). Instead, we propose Deep Improvement Supervision as a more general framework for guiding the intermediate steps of looped reasoning models.*

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 $30{\times}30$ puzzles.

Our DIS experiments adopt a minimal-compute design: we retain TRM's tiny 2-layer attention backbone and ARC

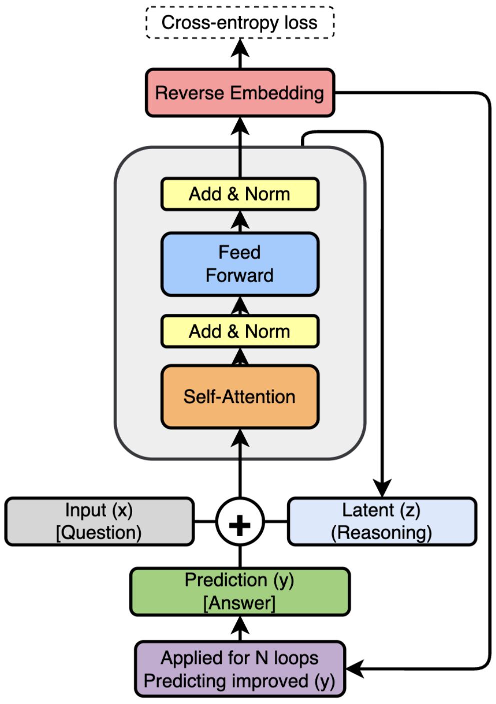

Figure 2. DIS model architecture. Algorithm starts with the embedded input question x, initial embedded answer $\mathbf{y}$, and latent state z. For up to n improvement steps, it tries to improve its answer $\mathbf{y}$ by simulating a discrete diffusion process, addressing any errors from its previous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

```
def latent_reasoning(x, y, z, n=2):
    with torch.no_grad():
        for j in range(T-1):
            for i in range(n):
                z = net(x, y, z)
            y = net(y, z)
    for i in range(n):
        z = net(x, y, z)
    y = net(y, z)
    return (y.detach(), z.detach()),
        output_head(y)

# Deep Improvement Supervision
for x_input, y_true in train_dataloader:
    y, z = y.init, z.init
    for step in range(N_supervision):
        y_step = f(x_true, y_true, step)
        x = input_embedding(x_input,
            step)
        (y, z), y_hat = latent_reasoning
            (x, y, z)
        loss = softmax_cross_entropy(
            y_hat, y_step)
        loss.backward()
        opt.step()
        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.

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`

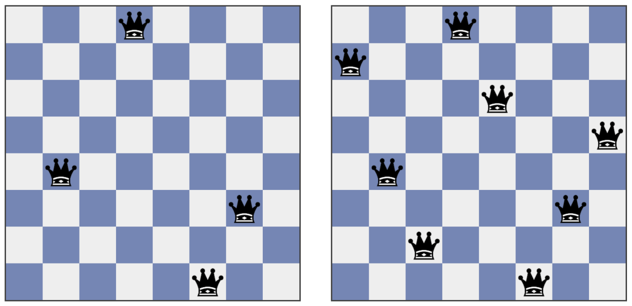

Figure 4. N-Queens reasoning problem example. Left is input and right is target solution.

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 examples with a seq. length of 64 and a vocabulary size of 3. For testing, we used 200 samples.

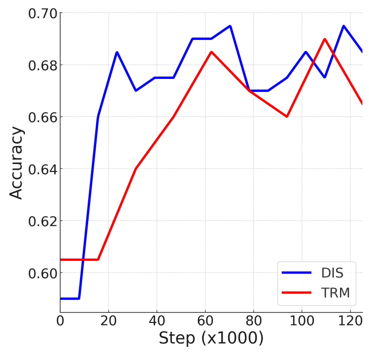

Figure 5. Accuracy curves on `N-Queens` problem.

Results. The aim of this experiment was to assess the impact of DIS training. As mentioned in Section 2.1, we used T = 1 and n = 2 for DIS, while the regular TRM of T = 3 and n = 6 were applied, with a halting head and 16 supervision steps. We achieved the same 0.69 accuracy, while using much fewer inference steps. The 0.8 mill parameters architectures was 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 so that our backbone and positional scheme match theirs. In evaluation, we apply the majority-vote of TRM over 1000 augmented inferences per puzzle.

5.3. ARC Model Settings

Objective. Each supervision step $s \in \{1, \ldots, 6\}$ 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 β_1=0.9, β_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_{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.

Test-time compute. We run the **same** N_{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

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	40.0	3.0
DIS	7M	40.0	3.0

For our experiments, we replicated the TRM experiments and achieved slightly lower results than those reported in the original paper. We also reimplemented TRM with the same hyperparameter settings as in our medium model. We decreased the number of latent and supervision steps to create the medium model same N_{sup}=16, T = 1, n = 2, and the halting mechanism remained active for TRM. In addition, we implemented a smaller network with reduced parameters to reproduce a compact model consisting of only 0.8 million parameters.

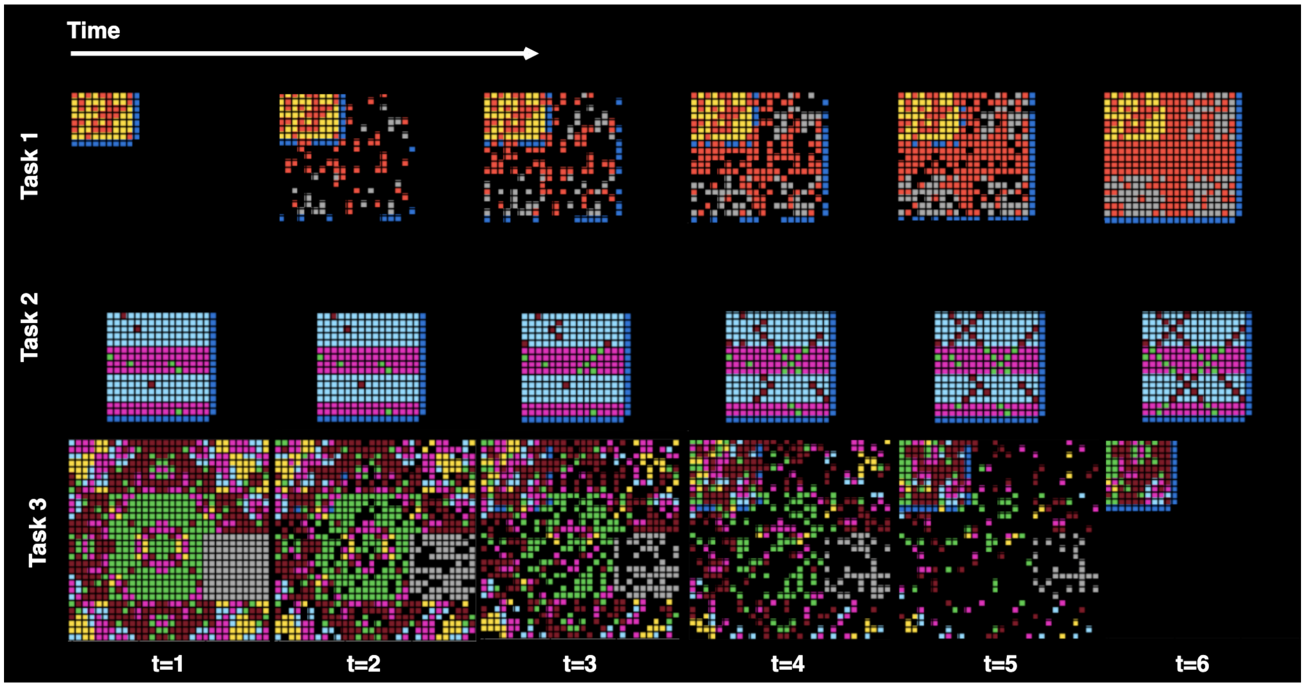

Figure 6. The linear corruption process is shown over six steps, from the initial input at time $t = 0$ to the target at time $t = 6$. A single training sample is illustrated per task.

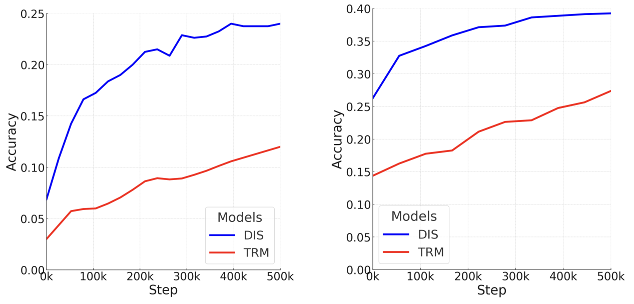

Figure 7. The DIS and TRM models pass@2 scores under the compact (left) and medium (right) setups.

The results are presented in Table 1 and Figure 7. As shown, for the compact model we dramatically outperform the original TRM. This shows that for TRM, latent reasoning steps 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 1.5 days.

6. Discussion

Regarding the guidance and corruption pipeline, a key improvement lies in optimizing the number of denoising steps. Currently, a fixed number of steps is used for different tasks, but it is evident that some tasks may benefit from more extensive denoising. This is supported by the original TRM paper, which highlighted the significant contribution of its halting mechanism to the final performance. Therefore, explicitly predicting the necessary number of denoising steps for each task could potentially enhance overall model performance. Another promising direction for technical improvement involves adopting a discrete latent space, which has been successfully used in deep learning models like the Dearmer model (Hafner et al., 2019). In general, the VQ-VAE's (Razavi et al., 2019) latent space has proven to be a robust and scalable representation for generation tasks.

In addition, as outlined in Section 4, there are several viable methods for generating intermediate steps. The discrete diffusion prior 6 represents one potential source for this choice, and our framework is designed to support various step-generation approaches. In our experiments, we also trialed the use of LLM-generated trajectories between transition samples and their targets. These trajectories were generated using the Gemini 2.5 Pro model. The model trained on these LLM-generated trajectories was a compact network with only 0.8 million parameters. However, we found that this method underperformed in comparison to the diffusion prior. We hypothesize that the LLM-generated trajectories do not provide a monotonic improvement path; the ”jumps” between intermediate steps can be highly nonlinear and difficult for a small model to capture. Consequently, a model trained with LLM improvement supervision achieved only 10% accuracy, compared to 24% achieved with diffusion prior.Exploring code-based generation of intermediate steps is 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.

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.

A. Analysis

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

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

We define the **Reasoning Residual** $\Delta\ell^s$ as the update vector proposed by the layer:

$$\Delta\ell^s = \ell_c^s - \ell_u^s$$

At test time, we define a guided policy π_w with guidance scale w. The probability of token a is given by the softmax of the guided logits:

$$\pi_w(a) = \frac{\exp(\ell_u^s[a] + w \cdot \Delta\ell^s[a])}{Z(w)}$$

where $Z(w) = \sum_{k\in\mathcal{V}} \exp(\ell_u^s[k] + w \cdot \Delta\ell^s[k])$. Let $\mathbf{y}$ be the index of the ground-truth correct token.

We first show the condition under which applying reasoning (increasing w) mathematically guarantees a reduction in loss.

Proposition A.1 (The Advantage Margin Condition)**.** *Let $\mathcal{L}(w) = -\log \pi_w(\mathbf{y})$ be the cross-entropy loss of the correct token $\mathbf{y}$. The loss strictly decreases as guidance increases ($\frac{d}{dw}\mathcal{L}(w) < 0$) if and only if:*

$$\Delta\ell[\mathbf{y}] > \mathbb{E}_{a\sim\pi_w}[\Delta\ell[a]]$$

Proof. We differentiate the loss function with respect to w step-by-step. Expand the loss definition.

$$\begin{aligned} \mathcal{L}(w) &= -\log\left(\frac{\exp(\ell_w[\mathbf{y}])}{Z(w)}\right) \\ &= -\ell_w[\mathbf{y}] + \log Z(w) \\ &= -(\ell_u[\mathbf{y}] + w\Delta\ell[\mathbf{y}]) + \log \sum_{k\in\mathcal{V}} \exp(\ell_u[k] + w\Delta\ell[k]) \end{aligned} \tag{16}$$

The derivative of the linear term $-(\ell_u[\mathbf{y}] + w\Delta\ell[\mathbf{y}])$ with respect to w is simply:

$$\frac{d}{dw}\big(-\ell_u[\mathbf{y}] - w\Delta\ell[\mathbf{y}]\big) = -\Delta\ell[\mathbf{y}]$$

We use the chain rule. Let $E_k = \exp(\ell_u[k] + w\Delta\ell[k])$. Then $Z(w) = \sum_k E_k$.

$$\begin{aligned} \frac{d}{dw}\log Z(w) &= \frac{1}{Z(w)} \cdot \frac{d}{dw} Z(w) \\ &= \frac{1}{Z(w)} \cdot \sum_k \frac{d}{dw}\exp(\ell_u[k] + w\Delta\ell[k]) \\ &= \frac{1}{Z(w)} \cdot \sum_k \exp(\ell_u[k] + w\Delta\ell[k]) \cdot \Delta\ell[k] \\ &= \sum_k \underbrace{\frac{\exp(\ell_w[k])}{Z(w)}}_{\pi_w(k)} \cdot \Delta\ell[k] \\ &= \mathbb{E}_{a\sim\pi_w}[\Delta\ell[a]] \end{aligned} \tag{17}$$

This result shows that the derivative of the log-normalizer is the expected value of the residual. Substituting Step 2 and Step 3 back into the loss gradient:

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

For improvement, we need the gradient to be negative ($\frac{d}{dw}\mathcal{L} < 0$):

$$\begin{aligned} -\Delta\ell[\mathbf{y}] + \mathbb{E}_{a\sim\pi_w}[\Delta\ell[a]] &< 0 \\ \Delta\ell[\mathbf{y}] &> \mathbb{E}_{a\sim\pi_w}[\Delta\ell[a]] \end{aligned} \tag{18}$$

□

This confirms that reasoning (the vector $\Delta\ell$) helps if and only if it pushes the correct answer $\mathbf{y}$ *harder* than it pushes the previous answer.

A.1. Global Convergence

While Proposition A.1 guarantees that increasing guidance improves the result locally, we must also prove that the entire sequential process converges to the correct solution $\mathbf{y}^\star$. We frame this using a Lyapunov stability argument.

Lemma A.2 (Discrete Lyapunov Contraction)**.** *Let $d(\hat{y}, \mathbf{y}^\star)$ be a bounded distance metric (e.g., Hamming distance) between a predicted sequence $\hat{y}$ and the ground truth $\mathbf{y}^\star$. Define the Lyapunov function (potential) at step s as $V_s = d(\operatorname{argmax}(\ell_c^s), \mathbf{y}^\star)$.*

Assume the following conditions:

1. ***Strictly Improving Targets:*** *The target generator Φ produces a sequence $\mathbf{y}_0^\dagger, \dots, \mathbf{y}_N^\dagger$ such that $d(\mathbf{y}_s^\dagger, \mathbf{y}^\star) \le d(\mathbf{y}_{s-1}^\dagger, \mathbf{y}^\star) - \delta$ for some strictly positive increment $\delta > 0$.*
2. ***Training Minimization:*** *The model minimizes the single-loss objective $\mathcal{L}_s = \mathrm{CE}(\ell_c^s, \mathbf{y}_s^\dagger)$ such that the output $\operatorname{argmax}(\ell_c^s) = \mathbf{y}_s^\dagger$ with high probability.*

Then, the system satisfies the Lyapunov stability criterion:

$$V_s - V_{s-1} \le -\delta < 0 \tag{19}$$

Proof. We examine the trajectory of the potential function V_s across the recurrent steps.

1. Connection to Supervision: By Condition 2 (Training Minimization), the model's conditional output at step s aligns with the target for that step:

$$\operatorname{argmax}(\ell_c^s) \approx \mathbf{y}_s^\dagger \tag{20}$$

2. Substitution into Potential: Substituting this into the definition of V_s:

$$V_s \approx d(\mathbf{y}_s^\dagger, \mathbf{y}^\star) \tag{21}$$

Similarly, for the previous step $s-1$:

$$V_{s-1} \approx d(\mathbf{y}_{s-1}^\dagger, \mathbf{y}^\star) \tag{22}$$

3. The Contraction: Using Condition 1 (Strictly Improving Targets), we substitute the inequality:

$$d(\mathbf{y}_s^\dagger, \mathbf{y}^\star) \le d(\mathbf{y}_{s-1}^\dagger, \mathbf{y}^\star) - \delta \tag{23}$$

Therefore:

$$V_s \le V_{s-1} - \delta \tag{24}$$

Conclusion: Since V_s is strictly decreasing and bounded below by 0 (when $\mathbf{y} = \mathbf{y}^\star$), the recurrent reasoning process is asymptotically stable and converges to the ground truth $\mathbf{y}^\star$ in finite steps $N \le \frac{V_0}{\delta}$. □

Lemma A.3 (Guaranteed Improvement)**.** *Assume the target generator Φ provides strictly improving targets, such that the likelihood ratio $\log \frac{P(\mathbf{y}_s^\dagger)}{P(\mathbf{y}_{s-1}^\dagger)} > 0$. Minimizing the sequential loss $\mathcal{L}_{DIS}$ drives the expected Advantage Margin to be positive:*

$$\mathbb{E}\big[\Delta\ell[\mathbf{y}^\star] - \mathbb{E}[\Delta\ell]\big] > 0. \tag{25}$$

Proof. We prove that minimizing $\mathcal{L}_{\text{DIS}}$ satisfies the Advantage Margin condition directly through substitution of the optimal solution.

1. Optimal Logits Assumption: We assume the TRM has sufficient capacity and has minimized $\mathcal{L}_{\text{DIS}}$ to convergence. This implies the conditional logits ℓ_c^s perfectly approximate the log-probabilities of the target distribution defined by $\mathbf{y}_s^\dagger$:

$$\ell_c^s \approx \log P(\cdot|\mathbf{y}_s^\dagger) + C_1 \tag{26}$$

By the recursive nature of the network (where the input to step s is the output of $s-1$), the reference logits ℓ_u^s approximate the previous target distribution:

$$\ell_u^s \approx \log P(\cdot|\mathbf{y}_{s-1}^\dagger) + C_2 \tag{27}$$

2. The Reasoning Residual: The residouble vector $\Delta\ell$ is the difference between these logits:

$$\begin{aligned}\Delta\ell &= \ell_c^s - \ell_u^s &(28)\\ &\approx \log P(\cdot|\mathbf{y}_s^\dagger) - \log P(\cdot|\mathbf{y}_{s-1}^\dagger) &(29)\\ &= \log \frac{P(\cdot|\mathbf{y}_s^\dagger)}{P(\cdot|\mathbf{y}_{s-1}^\dagger)} &(30)\end{aligned}$$

3. Evaluating the Margin for the Ground Truth $\mathbf{y}^\star$: We examine the first term of the Advantage Margin, $\Delta\ell[\mathbf{y}^\star]$:

$$\Delta\ell[\mathbf{y}^\star] \approx \log \frac{P(\mathbf{y}^\star|\mathbf{y}_s^\dagger)}{P(\mathbf{y}^\star|\mathbf{y}_{s-1}^\dagger)} \tag{31}$$

The lemma's premise states that the generator provides *strictly improving targets*. Mathematically, this means the likelihood of the ground truth $\mathbf{y}^\star$ increases with each step s:

$$\log \frac{P(\mathbf{y}^\star|\mathbf{y}_s^\dagger)}{P(\mathbf{y}^\star|\mathbf{y}_{s-1}^\dagger)} > 0 \tag{32}$$

Thus, $\Delta\ell[\mathbf{y}^\star]$ is strictly positive.

4. The Expectation Term: Assuming the targets are sharpening towards the correct solution (entropy decreases), the average residouble over all tokens $\mathbb{E}[\Delta\ell]$ is dominated by the probability mass shifting *into* $\mathbf{y}^\star$ and *out of* incorrect tokens. Therefore, the boost to the correct token dominates the average boost:

$$\Delta\ell[\mathbf{y}^\star] > \mathbb{E}[\Delta\ell] \tag{33}$$

This confirms that the expected Advantage Margin is positive. □