Deep Improvement Supervision

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

Abstract

Recent work has shown that small, iterative models such as Tiny Recursive Models can outperform large language models on complex reasoning problems such as those in the Abstraction and Reasoning Corpus (ARC). In our paper, we ask two simple questions: why are such models so efficient on the ARC-AGI problem, and how can they be improved? To answer these questions, we consider TRMs as implicit policy improvement algorithm. Based on this, we propose a new learning method that provides target for each model loop during the training process. In practice, we demonstrate that our approach makes training much more efficient. We avoid learnable halting mechanisms and simplify reasoning by reducing the total number of forward passes by 24x. We achieved the same quality as TRM with fewer resources. Also we achieved 24% accuracy on the ARC-1 problem with just 0.8 million parameters, outperforming most large language models.

1. Introduction

The use of iterative refinement loops on a model's outputs has become a major driver of progress in deep learning, 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 are achieved better performance than a standard one forward pass transformer on meta learning tasks, why utilizing 10x less number of parameters (Yang et al., 2023).

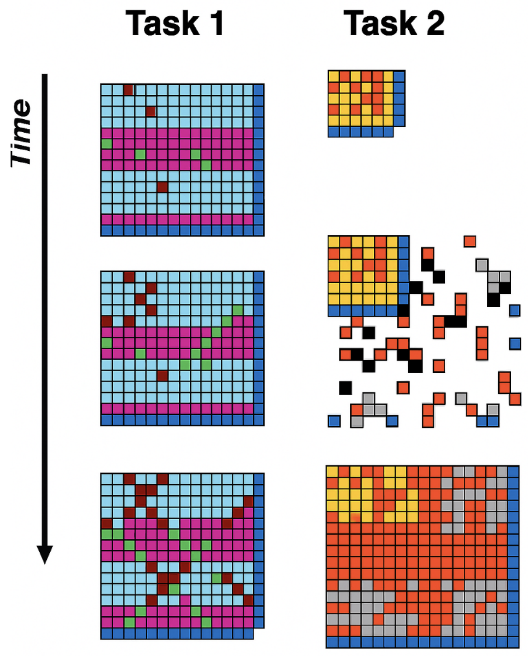

Figure 1. Blueprint of the discrete diffusion process on the ARC. Starting from the input x, following timestep t, we generate diffusion steps to the target y (Chollet, 2019; **?**).

Recently, models like Hierarchical Reasoning Models (HRM) (Wang et al., 2025) and Tiny Recursive Models (TRM) (Jolicoeur-Martineau, 2025), was build 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 the looped inference introduces a profound question. When a reasoning model executes a refinement step, what guarantees that it is on a path to a better answer? *How can we guide the model to ensure each step is verifiably closer to the target?*

In this paper, we propose a novel learning algorithm that modulate the iterative reasoning process to learn improvement itself as part of the optimization objective. Our main

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

Preprint. November 17, 2025.

finding is that guiding the model to predict intermediate steps (for example via discrete diffusion Fig. 1) during looped reasoning significantly reduces training complexity and improves generalization. To justify our findings, we considered TRM through the lens of reinforcement learning, demonstrating how a learned, guided self-improvement latent reasoning facilitates policy improvement beyond the policy given in data (Frans et al., 2025).

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 refinement steps, and 9x fewer latent reasoning steps. As a highlight of our method, our model achieves 24 percent 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 transformer repeatedly applies the same transformer block, with input injection each time, and is trained to make its intermediate loop outputs correct. This model was applied for the vairous tasks, showing the improvement over the single step models. 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 over 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 $\mathbf{z}_L, \mathbf{z}_H \in \mathbb{R}^{L\times D}$ across 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 prior work is $n = 2$ and $T = 2$. The final prediction is $\hat{\mathbf{y}} = \arg\max f_O(\mathbf{z}_H)$. During one forward pass, HRM evaluates the following updates:

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

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 through 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). 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, z_H is the state that the model reads out to produce the answer (the output head is applied to z_H: $\hat{y} = \arg\max f_O(z_H)$). z_L is a "working memory" state that's updated using the input x and the current answer, and is then used to update z_H. Because the loss is applied on the prediction from z_H, optimization pressure makes z_H look like (encode) the current solution. On the other hand, z_L is only supervised indirectly through its effect on z_H, so it's free to be an internal reasoning representation rather than a decodable solution. Within **same network** each recursion:

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

Prediction is made from z_H via the output head and trained with cross-entropy. This asymmetry (only z_H sees final z_L^{n+1}; and only z_H is decoded and penalized) naturally pushes z_H toward the space of valid answers, while z_L becomes the latent "reasoning scratchpad" that helps improve the next z_H. The TRM paper explicitly reframes this: "z_H is simply the current (embedded) solution... 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 z_H looks like the solved grid, while tokenized z_L looks like non-sensical tokens.

From the notation perspective, we want to note that TRM renames z_H to y (the running answer) and z_L to z (latent reasoning). The loop becomes: update z using (x, y, z); then update y using (y, z). Carrying both across deep-supervision steps lets the model iterate: z remembers how it got to the current guess (like a chain-of-thought), and y

stores the current guess itself. TRM trains a *single* halting probability via binary cross-entropy against correctness and removes the ACT *continue* head. It also uses an exponential moving average (EMA) of weights for stability on small data. Let $\mathcal{L}_{\text{task}} = \text{CE}(f_O(\mathbf{y}), \mathbf{y}_{\text{true}})$ be the prediction 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. One 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 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:

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

where $f : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a nonnegative, monotonically increasing function of advantage $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$, and $\hat{\pi}$ is a reference policy. According to the authors' formal proof 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 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 $o \in \{\emptyset, 1\}$ indicate *optimality*, whose defined via f as $p(o|s,a) = f(A(s,a))$, then the product policy from Eq. (3) can now be equivalently defined as: $\pi(a \mid s) \;\propto\; \hat{\pi}(a \mid s)p(o|s,a)$. As such, the score of the product policy above can be represented as the sum of two factors $\nabla_a \log \pi(a|s) = \nabla_a \log \hat{\pi}(a|s) + \nabla_a \log p(o|s,a)$. Then, using Bayes rule and w, CFG produces an attenuated score:

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

which corresponds to sampling from $\pi_w \propto \hat{\pi}(a \mid s)\, p(o{=}1 \mid s,a)^w$. 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 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 demonstrates policy improvement, yields consistently better returns than reference policy $\hat{\pi}$

3. Different Faces of Asymmetric Latent Reasoning

In this section, we decompose the elements of TRM to demonstrate its relationship with the well-established frameworks for generative modeling and reinforcement learning. The goal of this section is to provide a hypothesis on why TRM works and build a bridge to existing mathematical frameworks, which could be beneficial for further analysis and understanding of potential improvements.

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. The model uses two update modules, f_L^{ϕ} and f_H^{ϕ}, and an output head f_O. As established in §2.2, only $\mathbf{z}_H$ is decoded and penalized. We use an optimality variable $o \in \{\emptyset, 1\}$ to distinguish between unconditional ($o = \emptyset$) and conditional ($o = 1$) readouts.

3.1. Latent Reasoning as Classifier-Free Diffusion Guidance

The HRM model and further the TRM model are heavily based on latent reasoning processes. This reasoning process, as discussed in Sections 2.1 and 2.2, consists of two different-frequency reasoning loops. The HRM model uses two separate networks for the f_H (high) and f_L (low) frequency latents, resulting in two latents $\mathbf{z}_H$ and $\mathbf{z}_L$, respectively. The TRM replaces these two networks with a single one, showing better results and at the same time reducing the number of parameters by half. For clarity, we recall these two processes formally; let's consider a single latent reasoning cycle as:

One-cycle dynamics. $\mathbf{z}_L^{t+1} = f_L^{\phi}(\mathbf{z}_L^t, \mathbf{z}_H, x), \; t = 0, \dots, n-1, \; \mathbf{z}_L^o := \mathbf{z}_L^n, \; \mathbf{z}_H^+ = f_H^{\phi}(\mathbf{z}_H, \mathbf{z}_L^o)$.

This explicit asymmetry (only $\mathbf{z}_H$ is decoded and penalized; $\mathbf{z}_L$ is updated with x and conditions the next $\mathbf{z}_H$) allows us to define two distinct readouts. The ”unconditional” or ”reference” path is the output from the *current* state $\mathbf{z}_H$, while the ”conditional” path is the output from the *new* state $\mathbf{z}_H^+$ (which has seen the optimal latent $\mathbf{z}_L^o$). We define their respective logits using the output head f_O:

$$\textbf{Reference logits:}\quad \ell_{\mathrm{u}} := f_O(\mathbf{z}_L)$$
$$\textbf{Optimal logits:}\quad \ell_{\mathrm{c}} := f_O(\mathbf{z}_H^+) = f_O\big(f_H^{\phi}(\mathbf{z}_H, \mathbf{z}_L^o)\big)$$

Then, based on the classifier-free guidance formula, we can write down the final prediction of the TRM as a logit-space interpolation:

$$\ell_w = \ell_{\mathrm{u}} + w\big(\ell_{\mathrm{c}} - \ell_{\mathrm{u}}\big), \qquad \pi_w(\cdot \mid s) = \mathrm{softmax}\big(\ell_w\big). \quad (5)$$

Practical Note: *In the real TRM and HRM settings, this explicit combination does not happen. Vanilla TRM decodes directly from $\mathbf{z}_H^+$ (i.e., it only uses ℓ_c). With a rough assumption, we can say that TRM makes predictions equivalent to classifier-free guidance in an extremal case where the value w is very large, so the optimal case heavily outweighs the unconditional predictions. Equation (5) simply exposes a guidance knob w between ℓ_u and ℓ_c.*

We propose to explicitly use (5) as a last step in the inference. This can be included in a single line of code in the TRM, and our experiments show that such a modification does not harm results but brings latent reasoning into a well-established framework.

3.2. Latent Reasoning as Implicit Policy Improvement

In Section 3, we established that the asymmetric dynamics of TRM can be interpreted as a form of classifier-free guidance (CFG). We now demonstrate that this mechanism is not merely a heuristic but a practical implementation of a **controllable policy improvement operator**.

The core idea, formalized in (Frans et al., 2025), is to frame policy improvement as sampling from a ”product policy.” This policy is the product of a reference (or prior) policy, $\hat{\pi}(a|s)$, and an optimality function, f, that is conditional on the advantage $A_{\hat{\pi}}(s,a) = Q_{\hat{\pi}}(s,a) - V_{\hat{\pi}}(s)$.

$$\pi(a|s) \;\propto\; \hat{\pi}(a|s) f(A_{\hat{\pi}}(s,a)) \quad (6)$$

This formulation provides a direct path to provable policy improvement, which we can state formally.

Lemma 3.1 (Policy Improvement of Product Policies (Frans et al., 2025))**.** *If f is a non-negative, monotonically increasing function of the advantage $A_{\hat{\pi}}(s,a)$, then the product policy $\pi(a|s)$ defined in Equation (6) is a policy improvement over the reference policy $\hat{\pi}(a|s)$.*

See Theorem 1 in (Frans et al., 2025) for a formal proof. The intuition is that this product form up-weights actions with positive advantage and down-weights actions with negative advantage, increasing the expected return. A common and powerful choice for f is the exponentiated advantage, $f(A) = \exp(A/\beta)$, which is non-negative and monotonic. This choice recovers the KL-regularized policy objective.

Connecting Product Policies to TRM and CFG. The CFGRL framework connects this product policy to classifier-free guidance by defining the optimality function f as an optimality probability, $p(o = 1|s,a)$. We can further introduce an attenuation exponent w to control the strength of the improvement , which (Frans et al., 2025) proves also leads to monotonic improvement (Remark 2). This gives the precise form we concluded with in our CFG analysis:

$$\boxed{\pi_w(a \mid s) \;\propto\; \hat{\pi}(a \mid s)\, p(o{=}1 \mid s,a)^{w}} \quad (7)$$

The challenge is that $p(o = 1|s,a)$ is unknown. Instead of learning it explicitly, CFG provides a way to sample from π_w by training a single conditional model $\hat{\pi}(a|s,o)$ and applying Bayes’ rule.

$$\log \pi_w(a \mid s) \propto \log \hat{\pi}(a \mid s) + w \log p(o = 1|s,a) \quad (8)$$

Using Bayes’ rule, $\log p(o{=}1 \mid s,a) = \log \hat{\pi}(a \mid s, o{=}1) - \log \hat{\pi}(a \mid s) + C(s)$

$$\log \pi_w(a \mid s) \propto \log \hat{\pi}(a \mid s) + w\Big(\log \hat{\pi}(a \mid s, o = 1) - \log \hat{\pi}(a \mid s)\Big)$$

This final expression is exactly the score-space (or logit-space) combination of the ”unconditional” and ”conditional” policies we identified in Equation (5).

Thus, the asymmetric architecture of TRM is a natural way to learn the two components needed for this operation:

1. **The reference policy** $\hat{\pi}(a|s)$, derived from the unconditional logits $\ell_{\mathrm{u}} = f_O(\mathbf{z}_L)$.
2. **The optimal policy** $\hat{\pi}(a|s, o = 1)$, derived from the conditional logits $\ell_{\mathrm{c}} = f_O(\mathbf{z}_H^+)$.

This confirms that the TRM architecture naturally learns to perform iterative policy improvement steps via the latent memory $\mathbf{z}_L$. Because every recursion applies the f_L update that sees x before revising $\mathbf{z}_H$, the conditional path accumulates ”reasoning pushes” while the reference accumulates steps with that push absent. This makes the residual inherently different from 0 and, under the asymmetry-driven training signal, **monotone in cumulative advantage**—exactly what we need to conclude policy improvement.

4. Deep Improvement Supervision

Based on our previous section, we can assume that TRM architecture is capable of learning a policy-improvement factor $f(A_u^p)$ from only a final loss, this relies on inefficient, **long-term credit assignment.** We introducing a step-wise loss function with per-step targets, we provide a dense and direct supervisory signal. This reframes the learning problem from implicit advantage discovery (a hard RL problem) to explicit advantage imitation (a tractable SL problem). This "pre-defined advantage" makes the training of the $z_L \to z_H$ update (the core of the $\Delta\ell$ factor) significantly more stable and efficient, leading to a better and more reliable policy improvement at each step. Deep Improvement Supervision (DIS) augments the same backbone with *intermediate targets* $\{y_s^\dagger\}_{s=1}^{N_{\text{sup}}}$—constructed so that the expected discrepancy to $y^\star$ strictly decreases with s—and retains the standard CFG mixture 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

$$\mathbf{y}_1^\dagger,\ \mathbf{y}_2^\dagger,\ \ldots,\ \mathbf{y}_{N_{\text{sup}}}^\dagger \ =\ \Phi(\mathbf{x}, \mathbf{y}^\star;\ 1{:}N_{\text{sup}}), \tag{9}$$

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

$$\mathbf{y}_s^\dagger \ \sim\ q_{\beta_s}(\cdot \mid \mathbf{y}^\star), \qquad s = 1, \ldots, N_{\text{sup}}, \tag{10}$$

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 supervision step, detaching as in deep supervision:

$$\mathcal{L}_{\text{DIS}} \ =\ \sum_{s=1}^{N_{\text{sup}}} \mathrm{CE}\big(p_\theta(\cdot \mid \mathbf{x}, \mathbf{y}_s, \mathbf{z}_s, s),\ \mathbf{y}_s^\dagger\big)\,, \tag{11}$$

with optional increasing weights w_s to emphasize fidelity at later steps. The states $(\mathbf{y}_s, \mathbf{z}_s)$ are detached between the supervision steps, exactly as in deep supervision.

4.1. Algorithm

Our algorithm implements an iterative improvement training process where a model progressively refines its predictions. The system maintains both an answer state (y) and a latent reasoning state (z), which are updated through multiple neural network passes. 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. 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 multiple no-grad cycles before a grad cycle.
- **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 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 ARC-AGI dataset. Please see the appendix for additional experiments.

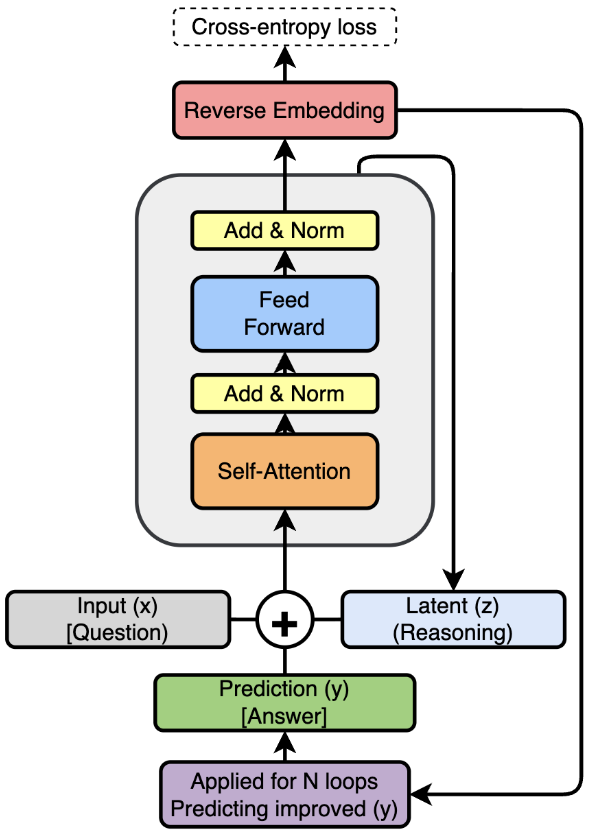

Figure 2. DIS model architecture. Algorithm starts with the embedded input question x, initial embedded answer y, and latent state z. For up to n improvement steps, it tries to improve its answer y by simulating a discrete diffusion process, addressing any errors from its previous answer in an parameter-efficient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

```
def latent_reasoning(x, y, z, n=2):
    with torch.no_grad():
        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

5.1. Model & Architecture

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"). This matches the micro-architecture TRM reports for ARC tasks (2 layers, attention, D=512) to isolate the contribution of DIS from capacity differences (Jolicoeur-Martineau, 2025). As in TRM, the model carries two states across supervision steps: the current solution $\mathbf{y}$ and a latent reasoning state $\mathbf{z}$. The same 2-layer network is called twice per internal step:

$$\mathbf{z} \leftarrow \mathrm{net}(\mathbf{x}, \mathbf{y}, \mathbf{z}), \qquad \mathbf{y} \leftarrow \mathrm{net}(\mathbf{y}, \mathbf{z}).$$

This "one-net, two-calls" design is exactly the simplification identified by TRM over HRM and is retained here (Jolicoeur-Martineau, 2025; Zhang & Sennrich, 2019; Shazeer, 2020).

5.2. Training Setup

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 the 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 TRM's hidden size D=512 **and call it medium model** in Table 1. When we are using D=256 and the single decoder layer model, resulting 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.

5.3. Datasets & Evaluation Protocol (ARC-AGI-1/2)

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 TRM's heavy augmentation pipeline 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. At evaluation, we apply TRM's majority-vote over 1000 augmented inferences per puzzle.

5.4. 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**
Direct prediction, small-sample training			
Direct pred	27M	21.0	0.0
HRM	27M	40.3	5.0
TRM-compact	0.8M	12.0	0.0
TRM-medium	7M	27.0	0.0
TRM	7M	40.0	3.0
DIS-compact (Ours)	0.8M	24.0	0.0
DIS-medium (Ours)	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 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. Additionally, we implemented a smaller network with reduced parameters to reproduce a compact model consisting of only 0.8 million parameters.

The results are presented in Table 1 and Figure 5. As shown, for the compact model we dramatically outperform the original TRM. This shows that, for the TRM, latent reasoning steps are important. We reduced the total number of latent steps by ninefold 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 comparable results 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 was shown to be 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 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. This makes training interpretable (we can audit failures at a specific s). DIS preserves TRM's compute profile per step (one gradient-bearing recursion cycle) and remains compatible with TRM's efficient halting head that avoids the extra forward pass required by HRM-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 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 in-

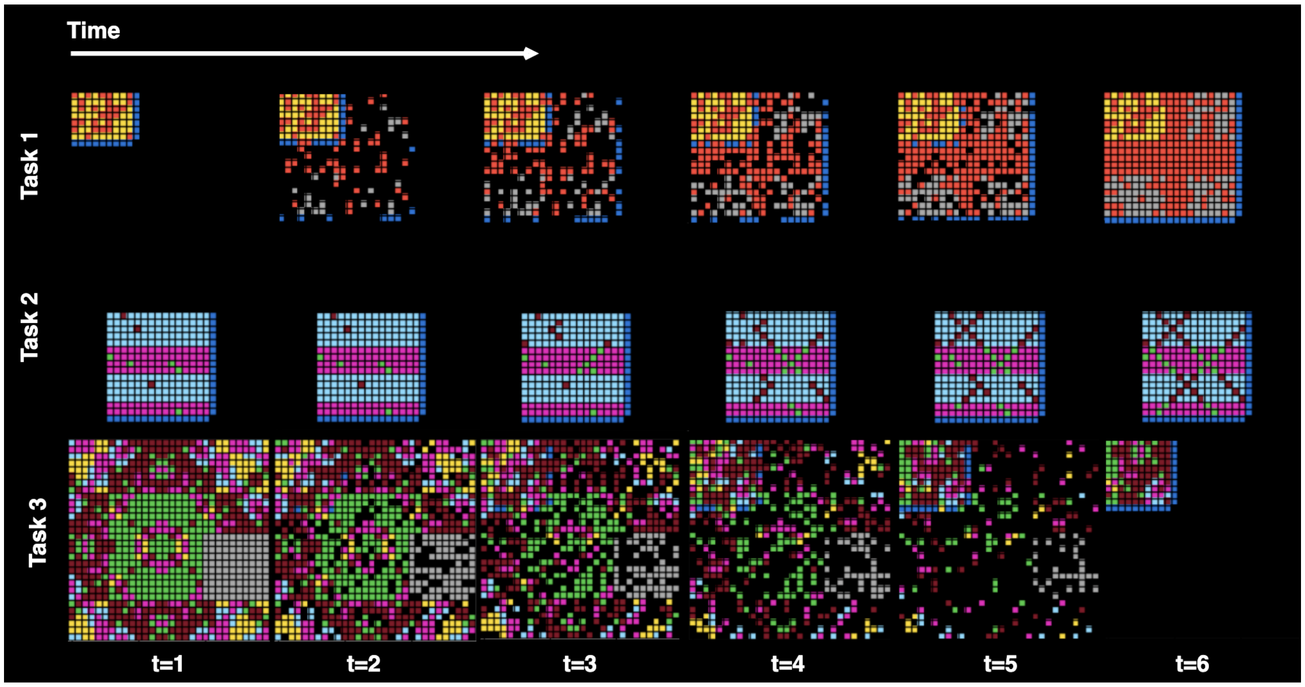

Figure 4. The linear corruption process is shown over six steps, from the initial input at time $t = 0$ to the target at time $t = 6$. A single training sample is illustrated per task.

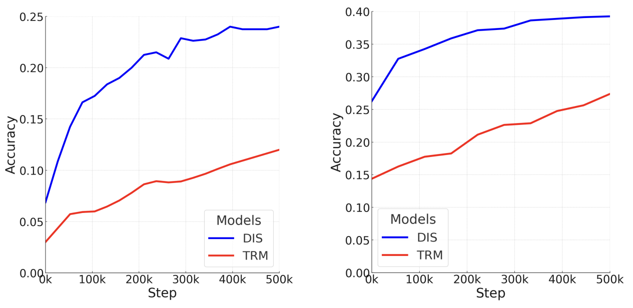

Figure 5. The DIS and TRM models pass@2 scores under the compact (left) and medium (right) setups.

volves adopting a discrete latent space, which has been successfully used in deep learning models like the Dearmer. In general, the VQ-VAE's latent space has proven to be a robust and scalable representation for generation tasks.

Also, as outlined in Section 4, there are several viable methods for generating intermediate steps. The discrete diffusion prior 4 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—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.

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.

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.

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.

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

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.

Zhang, B. and Sennrich, R. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019. arXiv:1910.07467.