Deep Improvement Supervision

Arip Asadulaev Rayan Banerjee Fakhri Karray Martin Takac
MBZUAI
`arip.asadulaev@mbzuai.ac.ae`

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 latent reasoning steps, reducing the total number of forward passes by 24x. Notably, we achieved 23% accuracy on the ARC-1 problem with just 0.8 million parameters, outperforming most large language models, and achieved the same quality as TRM with fewer resources.

1 Introduction

The application of multiple loops over a model's output has become a major driver of progress in deep learning. This idea has gained popularity in both small and large-scale models. Trillion parameters LLMs now highly rely on reasoning processes[3, 4] that involve refining their own output multiply time. Their primary mechanism for reasoning, Chain-of-Thought (CoT)[7] prompting, that relies on externalizing the reasoning process into a sequence of generated text. Furthermore, this approach often requires extensive training data and can be computationally expensive due to the large number of tokens generated for complex problems.

For smaller models, this idea is also proved itself, "looped transformers"[2, 8] 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[8].

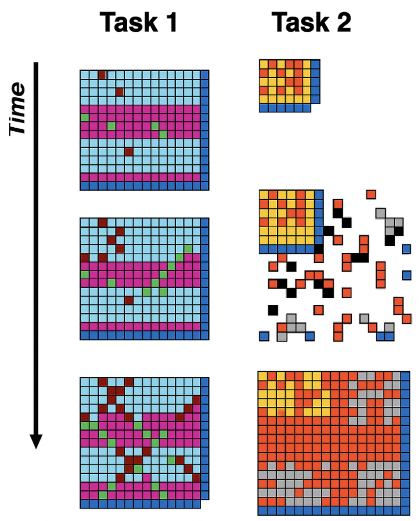

Figure 1: Blueprint of the discrete diffusion process on the ARC. Starting from the input x, following timestep t, we generate diffusion steps to the target y.

Recently, models like Hierarchical Reasoning Models (HRM) [6] and Tiny Recursive Models (TRM) [5], was build on the similar idea of reusing model output repeatedly in both the input and latent spaces. This models have demonstrated very impressive performance and even outperformed multi-billion parameter LLM models on complicated ARC-AGI 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?*

Preprint. Under review.

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 finding is that guiding the model to predict intermediate steps (for example via discrete diffusion Fig. 3) during looped reasoning significantly reduces training complexity and improves generalization. To justify our findings, we propose a theorem which, through the lens of reinforcement learning, demonstrates how a learned, guided self-improvement process facilitates policy improvement beyond data.

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. Avoiding training of the halting step module, 3x smaller number of refinement steps, and 9x smaller number of the latent reasoning steps. As a highlight of our method, our model achieves 23 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) 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 for every update of f_H 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:

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

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 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. Test-time evaluation runs a fixed maximum number of supervision steps to maximize accuracy.

2.2 Tiny Recursive Models

Tiny Recursive Models 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 each recursion:

- $z_L \leftarrow f_L(z_L + z_H + x)$ (so z_L directly sees the question x and the current answer),

- $z_H \leftarrow f_H(z_L + z_H)$ (so z_H is revised from z_L),
- prediction is made from z_H via the output head and trained with cross-entropy.

This asymmetry (only z_L sees x; 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 $\mathcal{L}_{\text{task}} = \mathrm{CE}(f_O(\mathbf{y}), \mathbf{y}_{\text{true}})$ 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. 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 Policy Improvement as Policy Product

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)[1]. 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:

$$\pi(a \mid s) \;\propto\; \hat{\pi}(a \mid s)\, f\big(A_{\hat{\pi}}(s,a)\big), \tag{2}$$

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)[1], 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.

3 TRM as Implicit Policy Improvement

In this section, we formally demonstrate that the proposed TRM mechanism functions as a policy improvement operator. By analyzing the asymmetry between the working memory (z_L) and the embedded solution (z_H), and exploiting the additive structure of the recurrence, we show that the TRM update effectively reweighs a reference distribution by a learned advantage factor. This provides a theoretical justification for the product-policy behavior without requiring changes to the inference procedure.

We map the TRM recurrence to a Reinforcement Learning step. Let the state at the recursion step s be denoted as $S_s = (x, y_s, z_s, s)$. Consistent with our architecture:

- $y_s \equiv z_H$ represents the *current embedded solution*. This is the only component decoded by the output head and directly penalized by the loss.
- $z_s \equiv z_L$ represents the *working memory* (latent reasoning). As noted in Section 2.2, only z_L directly receives the input injection x.

An action a is defined as a discrete edit produced from the solution embedding (e.g., the next token or grid update). The policy $\pi(\cdot \mid s)$ is parameterized by the output head f_O acting on z_H:

$$a \sim \pi(\cdot \mid s) = \mathrm{softmax}\big(f_O(z_H)\big). \tag{3}$$

The inner TRM step performs a two-stage update. First, the working memory absorbs the input x; second, the solution z_H is revised based on the memory:

$$\underbrace{z_L \leftarrow f_L(z_L + z_H + x)}_{\text{working memory sees } x} \quad \longrightarrow \quad \underbrace{z_H \leftarrow f_H(z_L + z_H)}_{\text{solution revised from } z_L} \quad \longrightarrow \quad \hat{y} = \arg\max f_O(z_H). \tag{4}$$

Crucially, the cross-entropy loss is applied only to the prediction derived from the updated z_H, creating an asymmetry where z_L is supervised indirectly through its ability to improve z_H.

3.1 Decomposition into Reference and Residual

To analyze the contribution of the TRM update, we construct a counterfactual *reference* distribution. We define the "unconditional" path as the path taken if the input injection x were clamped to zero for the current step. Note that this reference path is a theoretical construct for analysis and is not executed during inference.

Reference (Unconditional) Path:

$$z_L^{(u)} = f_L(z_L + z_H + 0), \quad z_H^{(u)} = f_H(z_L^{(u)} + z_H), \quad p_u(a \mid s) = \text{softmax}\big(f_O(z_H^{(u)})\big). \tag{5}$$

Actual TRM (Conditional) Path:

$$z_L^{(c)} = f_L(z_L + z_H + x), \quad z_H^{(c)} = f_H(z_L^{(c)} + z_H), \quad \pi(a \mid s) = \text{softmax}\big(f_O(z_H^{(c)})\big). \tag{6}$$

Let $\ell(a \mid z)$ denote the pre-softmax logit of the output head for action a. Because the recurrence utilizes additive updates in both stages, the influence of injecting x propagates to z_H as an additive perturbation. Assuming a linear (or locally linearized) readout f_O, the change in the hidden state results in an additive change in the logits. We define this residual contribution as:

$$\Delta\ell(s, a) \coloneqq \ell(a \mid z_H^{(c)}) - \ell(a \mid z_H^{(u)}). \tag{7}$$

Consequently, the logits of the actual TRM policy can be expressed as the sum of the reference logits and the residual:

$$\ell(\cdot \mid z_H^{(c)}) = \ell(\cdot \mid z_H^{(u)}) + \Delta\ell(\cdot). \tag{8}$$

3.2 Product-Policy Derivation

By substituting the decomposed logits into the softmax function, we derive the exact relationship between the actual policy π and the reference p_u:

$$\begin{aligned}
\pi(a \mid s) &= \frac{\exp\Big(\ell(a \mid z_H^{(u)}) + \Delta\ell(s, a)\Big)}{\sum_b \exp\Big(\ell(b \mid z_H^{(u)}) + \Delta\ell(s, b)\Big)} \\
&= \frac{\exp\big(\ell(a \mid z_H^{(u)})\big) \cdot \exp\big(\Delta\ell(s, a)\big)}{\sum_b \exp\big(\ell(b \mid z_H^{(u)})\big) \cdot \exp\big(\Delta\ell(s, b)\big)} \\
&= \underbrace{\frac{\exp\big(\ell(a \mid z_H^{(u)})\big)}{\sum_b \exp\big(\ell(b \mid z_H^{(u)})\big)}}_{p_u(a \mid s)} \cdot \frac{\exp\big(\Delta\ell(s, a)\big)}{\mathbb{E}_{b \sim p_u(\cdot \mid s)}\big[\exp\big(\Delta\ell(s, b)\big)\big]}
\end{aligned} \tag{9}$$

Thus, the TRM distribution is a product update of the reference:

$$\boxed{\pi(a \mid s) \propto p_u(a \mid s) \cdot f(s, a), \quad \text{where } f(s, a) \coloneqq \exp(\Delta\ell(s, a)).} \tag{10}$$

This result relies solely on the additive structure of the loop and the decoding of z_H. The term $f(s, a)$ acts as a latent indicator of improvement.

3.3 Alignment with Advantage and Policy Improvement

The asymmetry of the architecture ensures that the residual factor $f(s, a)$ aligns with the advantage function. Since only z_H is penalized, the only mechanism by which z_L can reduce the loss is by steering the update of z_H toward a better solution.

Let $A_{p_u}(s, a) = Q_{p_u}(s, a) - V_{p_u}(s)$ be the reference advantage, representing the improvement in the objective (e.g., reduced cross-entropy) achieved by action a. The indirect supervision of z_L implies a monotone coupling: the network learns to increase $\Delta\ell(s,a)$ for actions where $A_{p_u}(s,a)$ is positive. Therefore, $f(s,a)$ is a non-negative, non-decreasing function of the reference advantage, denoted as $f\big(A_{p_u}(s,a)\big)$. Substituting this into Eq. 10 yields:

$$\pi(a \mid s) \propto \underbrace{p_u(a \mid s)}_{\text{reference}} \cdot \underbrace{f\big(A_{p_u}(s,a)\big)}_{\text{improvement factor}} . \tag{11}$$

By the standard product-policy improvement theorem, reweighting a reference policy by a non-decreasing function of its advantage guarantees that π is a policy improvement over p_u.

We summarize this finding in the following proposition:

Remark 1:(TRM Step as Product-Policy Improvement) *Let $p_u(a \mid s)$ be the action distribution obtained by clamping the current input injection, and let $\pi(a \mid s)$ be the actual TRM distribution. If the readout logits are additive with respect to the injection effect, then:*

$$\pi(a \mid s) \propto p_u(a \mid s) \cdot \exp(\Delta\ell(s,a)). \tag{12}$$

Furthermore, if $\exp(\Delta\ell(s,a))$ *is a non-negative, non-decreasing function of the reference advantage $A_{p_u}(s,a)$, then π constitutes a strict policy improvement over p_u.*

This confirms that the TRM architecture naturally learns to perform iterative policy improvement steps via the latent memory z_L, without requiring explicit advantage estimation or modification of the sampling procedure at inference time. Moreover because every recursion applies the f_L update that sees x before revising y, the conditional path accumulates "reasonin 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

$$\mathbf{y}_1^\dagger,\ \mathbf{y}_2^\dagger,\ \ldots,\ \mathbf{y}_{N_{\text{sup}}}^\dagger \ =\ \Phi(\mathbf{x},\mathbf{y}^\star;\ 1{:}N_{\text{sup}}), \tag{13}$$

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), making the improvement direction 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

$$\mathbf{y}_s^\dagger \sim q_{\beta_s}(\cdot \mid \mathbf{y}^\star), \qquad s = 1, \ldots, N_{\text{sup}}, \tag{14}$$

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:

$$\mathcal{L}_{\text{DIS}} = \sum_{s=1}^{N_{\text{sup}}} \text{CE}\big(p_\theta(\cdot \mid \mathbf{x}, \mathbf{y}_s, \mathbf{z}_s, s),\ \mathbf{y}_s^\dagger\big)\,, \tag{15}$$

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

4.1 Algorithm

```
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 2: Pseudocode for reasoning with deep improvement supervision

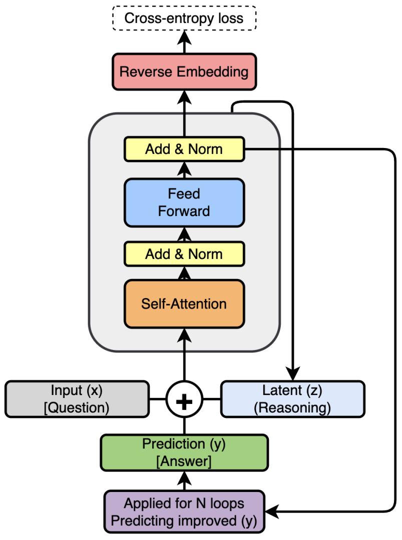

Figure 3: DIS model architecture. This method recursively improves its predicted answer y.

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**

- **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.

The DIS experiments are intentionally "minimal-compute": same tiny 2-layer attention backbone and ARC protocol as TRM, but **six** supervised improvement steps with **one** external cycle and **two** internal updates, and **no halting head**—allowing us to isolate the effect of **guided, monotone improvement**.

5 Experiments

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 [5]. 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 [5].

5.2 Training Setup

Objective. Each supervision step $s \in \{1, \dots, 6\}$ is trained toward a *step-specific intermediate target* $\mathbf{y}_s^\dagger$ produced by a discrete corruption schedule of 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 $\beta_1{=}0.9$, $\beta_2{=}0.95$, a 2k-step warm-up, and the stable-max cross-entropy variant for stability. For ARC experiments, we use weight decay 0.1 and we did not find EMA important. We match 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 $N_{\text{sup}}{=}6$ DIS steps. At each step we execute a single external cycle (since $T{=}1$) comprising two internal latent/answer updates ($n{=}2$), backpropagating through the full cycle; we then detach $(\mathbf{y}, \mathbf{z})$ before the next step. We **do not** train a halting/ACT head.

Test-time compute. We run the **same** $N_{\textbf{sup}}{=}6$ **steps** at evaluation, with fixed w and no sampling. 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].

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 $30{\times}30$. 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 [5].

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. We attach a puzzle-specific embedding token per augmented instance (and drop it for unconditional passes during CFG training).

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.

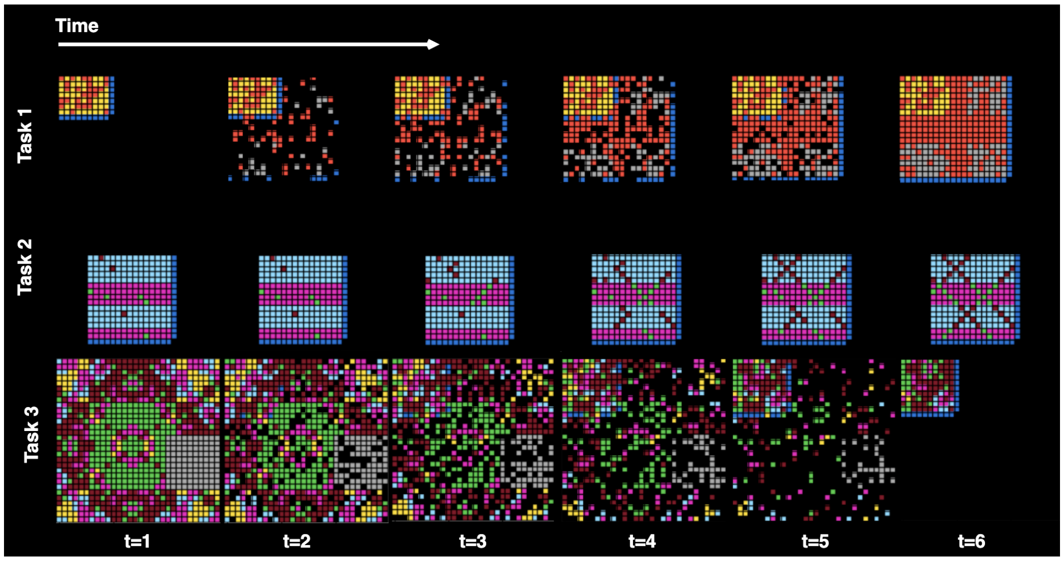

Figure 4: The corruption process is shown over six steps, from the initial input at time $t = 0$ to the target at time $t = 6$. A single training sample is illustrated per task.

5.4 Results

Table 1: Model Performance Comparison

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)	<u>0.8M</u>	<u>24.0</u>	0.0
DIS-medium (Ours)	7M	<u>40.0</u>	<u>3.0</u>

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 (no fixed-point assumptions, no two-network hierarchy), 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.

6 Conclusion

We demonstrates that small, iterative reasoning models can achieve competitive performance on complex reasoning tasks like the Abstraction and Reasoning Corpus, challenging the dominance of large-scale language models. By reinterpreting TRMs through the lens of reinforcement learning, the 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 via a discrete diffusion process, transforming the challenging problem of long-term credit assignment into a more tractable supervised learning task. 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 while maintaining accuracy.

References

[1] Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Diffusion guidance is a controllable policy improvement operator. *arXiv preprint arXiv:2505.23458*, 2025.

[2] Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pages 11398–11442. PMLR, 2023.

[3] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[4] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

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

[6] Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

[7] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

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