ONE STEP FORWARD AND K STEPS BACK: BETTER REASONING WITH DENOISING RECURSION MODELS

Chris Cameron, Wangzheng Wang, Nikita Ivanov, Ashmita Bhattacharyya,
Didier Chételat & Yingxue Zhang
Huawei Technologies
{chris.cameron,wangzheng.wang2,ashmita.bhattacharyya,}
{didier.chetelat,yingxue.zhang}@huawei.com

ABSTRACT

Looped transformers scale computational depth without increasing parameter count by repeatedly applying the same layer. However, training these models over long horizons creates significant optimization challenges. Specifically, it is difficult for looped transformers that start from noise to steer towards a complex output without additional supervision. Diffusion models tackle this issue by corrupting data with varying magnitudes of noise and training the model to reverse it in a single step. However, this process misaligns training and testing behaviour. We introduce Denoising Recursion Models, a method that similarly corrupts data with noise but trains the model to reverse the corruption over multiple recursive steps. This strategy provides a tractable curriculum of intermediate states, while better aligning training with testing and incentivizing non-greedy, forward-looking generation. Through extensive experiments, we show this approach outperforms the Tiny Recursion Model (TRM) on ARC-AGI, where it recently achieved breakthrough performance.

1 INTRODUCTION

Looped transformers, also known as recursive transformers, are a class of deep learning models that repeatedly apply the same transformer layer to an input to produce a prediction at test time (Giannou et al., 2023). Tying weights across loops imposes an architectural constraint that encourages the model to learn update rules that generalize across depth, acting as an implicit regularizer that can improve generalization. Empirically, this constraint is associated with stronger algorithmic reasoning (Dehghani et al., 2018; Geiping et al., 2025a) as well as improved parameter efficiency (Saunshi et al., 2025) and sample efficiency (Zhu et al., 2025b).

A novel looped transformer called the Tiny Recursion Model (TRM) (Jolicoeur-Martineau, 2025) recently made headlines by achieving breakthrough performance on ARC-AGI (Chollet et al., 2025), a notoriously difficult general reasoning benchmark consisting of novel puzzles, each with transformation that must be inferred from only a handful of examples. This accomplishment was particularly dramatic as this model, composed of barely seven million parameters, outperformed leading approaches based on Large Language Models (LLMs) such as o3-high (OpenAI, 2025), which has thousands of times more parameters and is pretrained at extreme computational cost.

The TRM is the direct successor of the Hierarchical Reasoning Model (HRM) (Wang et al., 2025) and the latest installment in a line of *iterative-refinement* looped transformers, dating back to Lee et al. (2018). Iterative-refinement models are non-autoregressive: at each loop they predict a complete, fixed-size output in parallel (one token/cell per position). Unlike chain-of-thought which appends newly generated tokens, these models are Markovian as subsequent loops rewrite the same-size prediction. They typically train by starting from noise and applying a sequence of recursions to match the target output. To borrow terminology from the *diffusion* literature, we could call these models *backward-training* as they start from noise and try to undo it, as in the backward process of a diffusion model.

TRMs substantially outperform standard transformers on ARC-AGI-2, however they still leave the majority of ARC-AGI-2 problems unsolved. One explanation is that it is too demanding for the model

to discover long, coherent trajectories starting from random noise and ending at highly-structured outputs without any additional supervision. This can be seen as analogous to sparse rewards in reinforcement learning (RL), where a random initial policy rarely stumbles onto a useful path to learn from. Long horizons becomes especially problematic because naively backpropagating through all recursive steps creates training instability (He et al., 2016) and requires storing activations across the entire unrolled computation. In practice, many looped models (Geiping et al., 2025a; Wang et al., 2025), including TRM, therefore rely on *Truncated Backpropagation Through Time* (TBPTT) (Williams and Zipser, 1995); they unroll the recursion for a fixed window of k steps, backpropagate through that window, then detach the state and continue the unrolling.

These long-horizon optimization issues motivate training objectives that avoid backpropagating through long unrolled recursions. Conditional diffusion models (Lai et al., 2025, Chapter 8) provide a clean workaround: they sample intermediate states from a process of corrupting the prediction target and train a *denoiser* model to reverse that corruption in a *single step*. This provides an automatic easy-to-hard curriculum by varying the degree of noise, considerably simplifying the optimization landscape. Perhaps backward-training, without any form of curriculum learning, is simply too demanding on the more difficult problems. Diffusion is also an iterative-refinement model since the same denoiser is applied repeatedly at test time, albeit diffusion has largely developed separately from the looped-transformer literature. Since they start from the target output and add noise to it, we could call these models *forward-training* (in reference to the forward process in diffusion).

While forward-training offers a solution to long-horizon training, we report experimental results in Section 5 showing that it substantially underperforms backward-training on ARC-AGI. We attribute this poor performance to a discrepancy between how the model is trained and how it is used at test time. During training, intermediate states are produced by noising the target and the model is optimized to remove that noise in a *single step*. At test time, however, the model is iterated on its own predictions for many steps; the intermediate states it encounters can therefore differ substantially from the noised-target states seen in training, and the one-step objective does not incentivize learning to produce intermediate states that remain useful under further self-application. Training through a recursive window gives the model the capacity for *implicit planning* (Guez et al., 2019) in latent space before committing to a final answer. This train-test mismatch could lead to compounding errors and encourage myopic, greedy updates when the model is unrolled at inference.

In contrast, backward-training methods possess the notable advantage of having no train-test discrepancy. This motivates the search for a way to directly incorporate the curriculum learning of forward-training while retaining the train-test alignment. Motivated by this idea, in this paper we propose Denoising Recursion Models (DRMs), a novel looped transformer that combines the strengths of TRMs and conditional diffusion models. At train time, this approach corrupts the ground truth with noise like in forward-training methods, but trains the model to recover the clean signal over a sequence of recursive steps like in backward-training methods, rather than a single jump. Crucially, our method only requires training over a single *window* of recursive steps, rather than long episodes with a TBPTT detachment schedule. At test time, a prediction is made as in both TRM and diffusion by repeatedly applying the transformer beginning from random noise. See Section 3 for the full details of the method.

In extensive experiments on ARC-AGI, using masking as the noise process, we found that DRMs outperformed TRMs at the same parameter count and surpassed the state of the art among open-source LLM-based baselines (NVARC, Sorokin and Puget (2025)) when controlling for training data. In addition, we investigate an alternative diffusion-TRM hybrid approach, which we call the State Perturbation Recursion Model (SPRM), where we inject noise into the intermediate latent states of the TRM procedure during training in order to explore more diverse trajectories. SPRM similarly improves upon TRM in small-data regimes, but it does not match DRM in larger-data regimes. In these data-rich settings, inherent data diversity likely makes the extra exploration from state perturbations unnecessary.

2 Related Work

Recurrent architectures are defined by the use of weight sharing to process sequences or generate computational depth. Historically, Recurrent Neural Networks (RNNs) (Elman, 1990) and LSTMs (Hochreiter and Schmidhuber, 1997) applied recurrence along the sequence dimension, updating

a fixed-size hidden state as they step through positions, which makes computation sequential and prevents parallelization across sequence positions. In contrast, looped transformers (Dehghani et al., 2018; Geiping et al., 2025a) apply recurrence along the depth dimension: they process the entire sequence in parallel while iterating a shared layer to scale depth arbitrarily. In this section we provide additional details on this prior work.

2.1 Looped Transformers

Dehghani et al. (2018) introduced the Universal Transformer, showing that recurrence across depth can improve reasoning while preserving parallelism across sequence positions. Subsequent work made this motivation more concrete in both theory and practice. Selsam et al. (2018) showed that looped graph networks trained on SAT can generalize to harder instances simply by deeper unrolling. Giannou et al. (2023) showed that looped transformers can emulate a small instruction-set computer, and Merrill and Sabharwal (2025) showed that these models can represent functions using only logarithmic depth in the number of steps required by discrete chain-of-thought. Empirically, looped models have shown improved parameter efficiency, sample efficiency, and length generalization on reasoning tasks (Saunshi et al., 2025; Geiping et al., 2025a; Fan et al., 2025; Zhu et al., 2025b). These results make looped transformers a natural architecture for algorithmic reasoning.

Long-horizon recursive training creates a common memory and optimization problem: storing activations scales linearly with unrolled depth, while gradients degrade as the horizon grows. Existing responses include TBPTT for explicit finite-horizon training (Williams and Zipser, 1995), one-step denoising objectives in diffusion, implicit differentiation in Deep Equilibrium Models (DEQ) (Bai et al., 2019; 2021), and RL-style optimization over long reasoning trajectories in autoregressive and diffusion settings (Black et al., 2024; He et al., 2025). We focus on a different point in this design space: preserving explicit multi-step differentiation through a short recursive window while avoiding the need to train from scratch over long trajectories.

Our work represents the latest in a line of research investigating *small* iterative-refinement recursion models, which have demonstrated remarkably data-efficient reasoning without massive pretraining, particularly ARC-AGI. See Appendix A.1 for a detailed overview of ARC-AGI and the most successful solutions. The Hierarchical Reasoning Model (HRM) (Wang et al., 2025) first demonstrated this potential by using alternating "fast" and "slow" recurrent steps, where the fast step is run many times for every execution of the slow step. Despite having only 27M parameters, HRM outperformed high-performance LLMs like o3-mini-high on ARC-AGI. This was simplified by the TRM (Jolicoeur-Martineau, 2025), which uses a single homogeneous recursive layer. Cutting down to just 7M parameters, TRM outperformed HRM and surpassed 100B+ parameter models like o3-high. Very recently, the Universal Reasoning Model (URM) (Gao et al., 2025) extended TRM by applying a 1D convolution over the token sequences at the end of the MLP block, yielding an $\sim$8% improvement on ARC-AGI-2. Hu et al. (2025) (ViARC) used a similarly small but non-looped vision transformer—a 2×2 patch encoding/decoding rather than encoding/decoding only at the token level—and improved performance over TRM. We think these TRM improvements of (1) changing encoding/decoding (ViARC) and (2) model architecture (URM) are fundamentally independent of our method; we anticipate that importing these ideas would see similar improvements.

2.2 Diffusion

Diffusion models are a type of looped, iterative-refinement model, first introduced by Sohl-Dickstein et al. (2015). They treat refinement as the reversal of a stochastic corruption process. Ho et al. (2020) introduced Denoising Diffusion Probabilistic Models (DDPM), which is the dominant formalism of diffusion today. While these foundations were established in continuous state-spaces, they can also be applied to tasks with discrete tokens. For discrete domains, this framework has been adapted via *stochastic transition matrices* (Austin et al., 2021) where tokens are corrupted by stochastically mapping tokens to a distribution over the vocabulary, or *absorbing masking states* where the models learns to iteratively recover tokens from a masked input (Chang et al., 2022). Recent methods like Llada (Nie et al., 2025; You et al., 2025; Zhu et al., 2025a) have shown that masked diffusion for training language models can be competitive with autoregressive models.

The question of whether realistic intermediate states can be represented as simple corruptions of the target also arises outside our setting. GFlowNets (Bengio et al., 2021), for example, model probability

flow through structured state trajectories when intermediate states must obey combinatorial or causal constraints. The key difference is that GFlowNets assume an explicit discrete state space and transition rules, whereas our intermediate reasoning states are latent continuous representations learned end-to-end.

2.3 Intersection of Diffusion and Recursion

Lee et al. (2018) introduced an early iterative-refinement model and trained it using a *denoising-autoencoder* objective. In our terminology, their procedure stochastically interleaves forward-training (denoising a corrupted version of the target) and backward-training (refining from an uninformative initialization), sampling between the two modes with a mixture probability. Although this work predates diffusion terminology, the denoising-autoencoder component plays a similar role to diffusion by providing intermediate difficulty states via controlled corruption of the ground truth. Our DRMs are closely related but differ in how these ingredients are combined. Lee et al.'s forward-training updates supervise a single refinement step, whereas their backward-training updates still require traversing the trajectory from scratch. DRMs instead couple forward corruption with multi-step refinement within each example: we sample a corruption level, initialize the recursion at the corrupted target, and train to recover the clean output over a window of k recursive steps.

Several recent papers explicitly frame looped models as diffusion-like processes. Geiping et al. (2025b) build on Chen et al. (2024)'s work that proposed an asynchronous update schedule where the model processes token k at loop depth $i - k$, effectively creating a *diagonal frontier*. Unlike our methods, their primary focus is computational efficiency during inference and their experiments use pretrained looped models (Geiping et al., 2025a) that do not use diffusion objectives, whereas our approach is a training method. Similarly, Zhou et al. (2025) characterize looped models as continuous diffusion processes lacking intermediate noise and they propose a hybrid "co-evolutionary" architecture that combines discrete and continuous denoising.

An alternative to the backward-training methods used in standard iterative-refinement models is to optimize the denoising trajectory with reinforcement learning (RL) (Black et al., 2024). He et al. (2025) recently applied this idea to masked diffusion, using intermediate rewards based on similarity to the ground truth. These rewards are analogous in spirit to the deep-supervision signals used in TRM, but the credit-assignment story is different: because the denoising policy is optimized over the full rollout, each update is valued according to how it contributes to eventual success under the same multi-step process used at inference. In this sense, RL directly aligns the training objective with test-time denoising, and He et al. (2025) report substantial improvements over LLaDA on mathematical and reasoning benchmarks.

A related idea appears in a different RL setting: recurrent policies acting in external MDPs rather than denoising trajectories. There, repeated internal computation has been discussed as a form of *implicit planning*, where the policy can refine its latent state before committing to an action. Guez et al. (2019) introduced the Deep Repeated ConvLSTM (DRC), and Bush et al. (2025) later provided evidence consistent with planning-like latent updates in this architecture. The analogy to DRMs is only partial, since we train with supervised losses rather than policy optimization. Still, backpropagating through multiple recursive denoising steps gives the model a related incentive to shape latent states for their downstream usefulness, rather than only for immediate denoising gain.

3 Methods

3.1 Preliminaries: Training Looped Models

In this work, we operate in the non-autoregressive sequence-to-sequence setting. We are given a dataset of training pairs $(X, Y) \sim \mathcal{D}$, where the input X and ground truth Y are sequences of length M. Both are represented as one-hot encodings over a vocabulary $\mathcal{V}$, such that $X, Y \in \{0, 1\}^{M \times |\mathcal{V}|}$. Each unique token is mapped to a learned continuous embedding in $\mathbb{R}^d$. To generate predictions from a latent state $H \in \mathbb{R}^{M \times d}$, the model uses a decoder head to generate a probability distribution over the target tokens:

$$P_\theta(Y \mid H) = \text{softmax}(\text{dec}(H)). \tag{1}$$

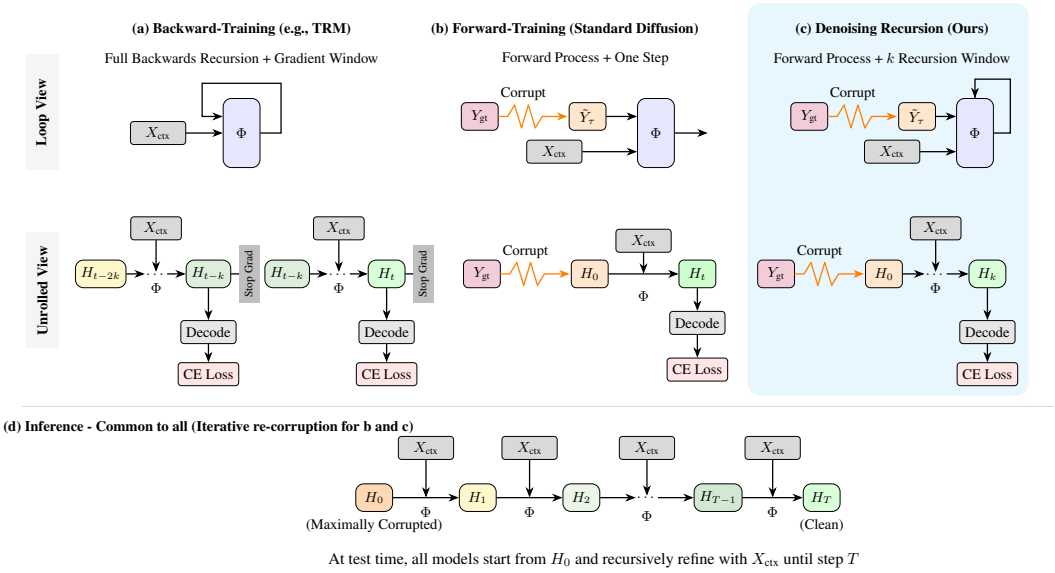

Figure 1: Architectural comparison between (a) backward-training (e.g., TRM), (b) forward-training (Standard Diffusion), and (c) DRMs. Like forward-training, DRMs sample the forward process and like backward-training, they train over k recursions of Φ. All methods recursively map from maximal corruption to a clean target during inference (d).

We focus on *looped models*, a class of functions where the latent state is generated via a shared transition operator Φ_θ applied recursively:

$$H_t = \Phi_\theta(H_{t-1}, X, \dots), \quad t = 1 \dots T \tag{2}$$

where H_t is the latent state at recursion step t and H_0 is an initialization distribution (typically Gaussian noise). T represents the maximum recursive depth used during inference. While all looped models share this inference structure, they have different ways of training Φ_θ to manage the training instability and memory constraints over a long horizon T.

3.1.1 Forward-Training Models (Diffusion)

Diffusion models decouple the recursive dependency during training. A forward process corrupts the ground truth Y into a noisy state $\tilde{Y}_\tau$ according to a schedule of timesteps $\tau \sim \mathcal{U}(1 \dots T)$, effectively fast-forwarding the recursion process to intermediate states of varying levels of completeness. Let $q_\tau(\tilde{Y} \mid Y)$ denote the forward corruption process. The model is trained to reverse this corruption in a *single step*; Φ_θ is applied exactly once to the noisy input $\tilde{Y}_\tau$, preventing gradients from flowing through time. We minimize the following reconstruction loss across noise levels τ:

$$\mathcal{L}_{\text{forward}}(\theta) = \mathbb{E}_\tau \left[-\log P_\theta(Y \mid \Phi_\theta(\tilde{Y}_\tau, X, \tau)) \right]. \tag{3}$$

3.1.2 Backward-Training Models

Backward-training models (like TRM) are trained starting from pure noise and unrolling the recursion. Over long enough horizons and difficult tasks, these methods must use some form of *Truncated Backpropagation Through Time* (TBPTT). The common instantiation of this, as is done in TRM, is to partition the full trajectory T into intervals of size k. The gradients are computed within a window, but the recurrent state is detached from the computation graph at the boundaries. Formally, let $\text{sg}(\cdot)$ denote the stop-gradient operator. The state update is:

$$\tilde{H}_t = \begin{cases} \text{sg}(H_t) & \text{if } t \bmod k = 0 \\ H_t & \text{otherwise} \end{cases} \tag{4}$$

The recursion continues as $H_{t+1} = \Phi_\theta(\tilde{H}_t, X, \dots)$. Supervision is applied at the end of each interval k up to the maximum depth T:

$$\mathcal{L}_{\text{backward}}(\theta) = \sum_{i=1}^{T/k} -\log P_\theta(Y \mid \tilde{H}_{i \cdot k}). \tag{5}$$

Unlike diffusion, the state $\tilde{H}_{i \cdot k}$ is the result of applying Φ_θ sequentially, allowing the model to learn forward-looking algorithmic operations while maintaining tractable training dynamics via the stop-gradient operator.

3.2 Denoising Recursion Models (DRMs)

DRMs hybridize forward-training and backward-training looped transformers. We leverage the "fast-forward" capability of diffusion to initialize the model at an arbitrary difficulty level (noise scale τ), but we then allow the model to resolve this state over a recursive window of k steps. We initialize the embedded corrupted target: $H_0 = \text{Embed}(\tilde{Y}_\tau)$ where $\tilde{Y}_\tau$ is sampled from a forward diffusion process $q_\tau(\tilde{Y} \mid Y)$. We then unroll the transition operator Φ_θ for a fixed window of k iterations and minimize the negative log-likelihood of the clean ground truth Y given the resulting state H_k:

$$\mathcal{L}_{\text{DRM}}(\theta) = \mathbb{E}_\tau \left[-\log P_\theta(Y \mid H_k) \right]. \tag{6}$$

By initializing with $\tilde{Y}_\tau$, we retain the state diversity of diffusion, exposing the model to a vast range of intermediate states without needing to generate them. And by backpropagating through k steps of Φ_θ, we reintroduce the forward-looking inductive bias of recurrent models, training the network to perform multi-step algorithmic refinement rather than greedy denoising. See Figure 1 for an illustrative comparison. At inference, DRMs begin from the maximally corrupted state supported by the forward process and iteratively apply the denoiser along a decreasing noise schedule until a clean prediction is produced. In the masked discrete case, this corresponds to initializing all output tokens as `MASK` and repeatedly predicting a full grid followed by partial re-masking.

3.3 State Perturbation Recursion Models (SPRMs)

While DRMs start with a forward step, followed by backward steps, we can also consider a hybrid method with alternating backward and forward processes. We call this State Perturbation Recursion Models (SPRMs) because it forces the model to continuously recover from stochastic perturbations along states of its own reasoning path. Rather than denoising from a noisy ground truth, SPRMs effectively denoise from its own incumbent solution. This increases the diversity of intermediate states, which could improve robustness at test time when encountering out-of-distribution states. Let $q_\tau(\tilde{H} \mid H)$ denote a forward corruption kernel (a conditional distribution over perturbed states) parameterized by a noise level τ—e.g., DDPM (Ho et al., 2020). At each recursion step t, we sample a noise level τ_t and perturb the previous state via the forward process:

$$\tau_t \sim \mathcal{U}(1, \dots, T), \qquad \tilde{H}_{t-1} \sim q_{\tau_t}(\cdot \mid H_{t-1}), \qquad H_t = \Phi_\theta(\tilde{H}_{t-1}, X). \tag{7}$$

This corruption forces the operator Φ_θ to map from the noisy latent manifold back to the solution manifold at every iteration. We think continuous state perturbations could discourage the model from converging to brittle trajectories and better encourage a robust attractor field around correct reasoning trajectories.

4 Experimental Setup

4.1 Data

Each ARC task provides a small set of input-output training examples and 1–3 held-out test inputs, where each grid is at most 30×30 and each cell takes one of 10 colours; we report pass@2 exact-match accuracy on the held-out outputs. We evaluate on two datasets: ARC2-Public Eval (120 difficult tasks) and ARC2-Public Train, which we refer to as *ARC-Easy* (1000 generally easier tasks). For pretraining, we use three settings of increasing scale: ARC-Easy + ConceptARC (Moskvichev et al., 2023), reARC (Hodel, 2024), and NVARC-Train (Sorokin and Puget, 2025). We exclude NVARC-Eval because it is derived from ARC2-Public Eval and would introduce data leakage. Dataset sizes are summarized in Table 1.

4.2 IMPLEMENTATION

Our method is architecture agnostic, so to isolate the effect of the training method we use the TRM architecture and hyperparameters of Jolicoeur-Martineau (2025) (Table 3). TRM is a backward-trained looped transformer with four additions: a learned **task embedding** concatenated at each recursion, which shares information across examples within a task; a **scratchpad** state split (y_t, z_t) with latent z_t; **warm-up loops** before backpropagation; and a scalar **Q-head** that predicts exact-match correctness for early exit and selecting a final solution across augmentations. See Figure 7 for pseudocode.

4.2.1 DENOISING RECURSION MODEL

Given that we tokenize each cell in ARC grids, we employ a *discrete masking schedule*—an established technique in language diffusion (Nie et al., 2025)—rather than adding Gaussian noise to continuous embeddings[1]. We used a cosine noise schedule, observing substantially faster training convergence compared to a linear schedule. This aligns with prior work on discrete diffusion (Chang et al., 2022) and continuous diffusion (Nichol and Dhariwal, 2021). Linear schedules tend to destroy information too often, while cosine schedules smooth out the rate of information destruction. Let $s \in [0, 1]$ represent the normalized diffusion timestep. We compute the signal retention rate $\bar{\alpha}_s$ as:

$$\bar{\alpha}_s = \cos^2\left(\frac{\pi s}{2}\right). \tag{8}$$

The fraction of tokens to be masked at step s is given by $r_s = 1 - \bar{\alpha}_s$. For a diffusion timestep $s \sim \mathcal{U}[0, 1]$, we select $\lfloor r_{s_{i-1}} \cdot M \rfloor$ indices uniformly at random[2] and replaced the tokens at those indices with a special `MASK` token. We used separate token embeddings for mapping the masked ground truth into latent space than those used for input grids. Since TRM adds the input embedding to the output embedding, we hypothesized these separate embeddings would help to avoid interference. See Figure 7b for pseudocode.

During inference, we reverse this process using an iterative "generate-and-remask" strategy as in Nie et al. (2025) except we do not freeze tokens in place if they are not selected to be remasked. We define a sequence of discrete timesteps $s_T, \ldots, s_0$ decreasing from 1 to 0. We initialize the grid as fully masked. At each iteration i, the model predicts logits for all grid cells given the current input. We first decode a fully populated candidate grid $\hat{y}_0$ by taking the argmax of the logits at every position. To construct the input for the subsequent step $i - 1$, we calculate the required number of masked tokens $k = \lfloor r_{s_{i-1}} \cdot M \rfloor$ based on the schedule. We then select k indices *uniformly at random* to be re-masked as in training, retaining the predicted values from $\hat{y}_0$ at the unselected indices. This process—predicting the full grid and stochastically re-masking a subset of cells—repeats until $s_0 = 0$, at which point no masks remain. We tried confidence-based remasking—i.e., masking lowest confidence tokens (Nie et al., 2025)—but did not observe any performance improvement.

4.2.2 STATE PERTURBATION RECURSION MODEL

For our secondary hybrid method, we inject noise directly into the hidden states rather than the input tokens. This perturbation is applied exclusively at the truncation points ("outer loops" of TRM). We adopt the standard linear variance schedule from Denoising Diffusion Probabilistic Models (DDPM) (Ho et al., 2020). We define a variance schedule β_t that increases linearly from $\beta_{\text{start}} = 10^{-4}$ to $\beta_{\text{end}} = 0.02$. At the end of each truncated backpropagation window (every k steps), we perturb the hidden state $H_{i \cdot k}$ before passing it to the next window. We use the variance-preserving update:

$$H_{\text{next}} = \sqrt{1 - \beta_\tau} H_{\text{prev}} + \sqrt{\beta_\tau}\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}), \tau \sim \mathcal{U}(1 \ldots T). \tag{9}$$

We do not add noise during inference. We tried adding noise during inference according to the same DDPM noise schedule used during training and observed minor performance drops.

[1] We first tried standard DDPM on the grid embeddings but observed very poor performance. We suspect this was because the model strategically learned embeddings that were recoverable from noise.

[2] Uniform at random can create a "salt and pepper" masking that we thought could be quite easy to interpolate. We tried sampling in a spatially correlated way by using a 2D Gaussian kernel but we did not observe any benefit.

4.2.3 BASELINES

Standard Diffusion and Transformer Baselines. To isolate the necessity of the recursive inductive bias, we evaluated standard architectures that match the computational budget of the TRM but lack the recurrent weight sharing. We evaluated standard diffusion models using the exact same discrete masking schedule and inference procedure as the DRM implementation described in Section 4.2.1. The 7M model matches the capacity of the DRM transition operator. The 70M model is constructed by stacking transformer blocks. to match the total computational depth (FLOPs) used by the TRM during a single backpropagation window (which consists of four recurrent steps plus one final processing loop). We also evaluated a standard Transformer with the same stacked architecture. To determine if recursive layers are required specifically within a gradient block, we also evaluated the repeated application of a stacked transformer.

TRM without deep supervision. We include a TRM baseline where we trained solely on the loss at the final step of the recursion window, removing the intermediate truncated backpropagation losses. This serves to measure the dependence of the architecture on dense supervision signals.

State-of-the-Art (NVARC). We compare our results against the current state-of-the-art (among open-source models) NVARC system (Sorokin and Puget, 2025), which achieved first place in the 2025 ARC Challenge. We evaluated their 4-billion parameter LLM while controlling for the dataset.

4.3 TRAINING AND EVALUATION

We follow the standard TRM protocol of Jolicoeur-Martineau (2025) unless specified otherwise. We did all exploratory hyperparameter tuning on ARC2-Easy to avoid overfitting ARC2-Eval. See the appendix for specific training (Table 2) and architecture parameters (Table 3).

Data Preparation, Augmentation, Grid Selection. For each ARC task, we sample 1000 augmentations. For each augmentation we (1) randomly permute the colours, (2) select a random transformation (e.g., flip, rotate) and (3) if the grid is smaller than 30×30, randomly translate each pair of grids into a fixed 30×30 template (background cells outside the original grid are masked when loss is measured). We learned embedding vectors for the 10 colours and the `MASK` token. We represent each ARC grid as a sequence flattened in row-major order and use Rotary Positional Embeddings (RoPE) (Su et al., 2024). At evaluation, we group identical predictions across augmentations, average their predicted q-values, and select the top-k unique outputs; Table 1 reports pass@2.

Pretraining and Finetuning. For the large-scale data settings, we used a two-stage process: we pretrained the model on reARC for 100K epochs and on NVARC+reARC for 2K epochs, using identical hardware for both. Following pretraining, we performed a finetuning stage of 300K epochs, where we strictly followed the TRM methodology of jointly training on the provided training pairs of both ARC2-Train and ARC2-Eval tasks. Evaluation is performed on the held-out test pairs of the ARC2-Eval tasks at termination.

NVARC Baseline Details. For the NVARC baseline, we pretrained until the validation loss plateaued (approx. 2K epochs). We used Sorokin and Puget (2025)'s test-time training implementation for finetuning to ARC2-Easy and ARC2-Eval.

5 RESULTS

The Necessity of Recursion. We first establish the baseline difficulty of the task. We find that standard non-recursive architectures are insufficient for ARC-AGI. A standard transformer achieved only 25.5% on ARC-Easy, far below the recursive TRM baseline of 45.7%. Crucially, we found that diffusion without backpropagating through multiple recursive steps performs very poorly. A standard diffusion model (matching the DRM architecture but without looping) failed completely, achieving $\sim$0% accuracy. Even when we stacked layers to match the compute budget of recursion (70M params), we still could not get above 1%. This suggests that iterative refinement via shared weights is the primary driver of performance.

DRM Stabilizes Training without Deep Supervision. Standard TRMs rely heavily on deep supervision (TBPTT) to guide intermediate states. When we removed this supervision ("TRM w/o deep sup"), performance fell by 11% (from 45.7% to 34.7%) on ARC-Easy. In contrast, our DRM

Table 1: Pass@2 exact-match accuracy results on held-out data for every task comparing **DRM** and **SPRM)** methods against the TRM baseline and state-of-the-art NVARC. The 7M models use identical architectures. The 14M settings increased embedding size from 512 to 768, number of heads from 8 to 12, and unroll length for backprop from 4 to 6.

Method	#Params	ARC-Easy	ARC2-Eval (by Pretraining Data)		
		(No Pretrain)	+ARC-Easy	+reARC	+NVARC Train
		-	*1k tasks (3.5 ex/task)*	*400 tasks (1k ex/task)*	*47k tasks (24 ex/task)*
Baselines					
Transformer	7M	10.9	-	-	-
Transformer	70M	25.5	1.7	-	-
Transformer *w/ deep sup*	70M	37.2	2.5	-	-
Masked Diffusion (Standard)	7M	0.0	0.0	-	-
Masked Diffusion (Standard)	70M	0.7	0.0	-	-
TRM *w/o deep sup*	7M	34.7	3.3	-	-
TRM *w/o inner loop*	34M	39.8	3.0	-	-
LLM (NVARC)	4B	43.2	**13.8**	15.6	22.6
TRM (Baseline)	7M	45.7	6.3	12.4	12.5
SPRM (Ours)		**50.7**	**10.1**	11.8	14.4
DRM (Ours)		50.5	9.6	**14.7**	**16.7**
TRM (Baseline)	14M	53.2	10.6	18.8	21.8
SPRM (Ours)		**55.4**	11.8	17.4	22.2
DRM (Ours)		55.0	**13.3**	**21.0**	**24.9**

method trains without any intermediate gradient truncation or deep supervision yet outperformed the TRM baseline by nearly 5% on ARC-Easy (50.5% vs 45.7%). This suggests that the diffusion objective provides a useful curriculum for learning long-horizon reasoning trajectories.

DRM Outperforms TRM. These gains transfer to the ARC2-Eval benchmark, which consists of tasks with higher difficulty. On the base setting (no pretraining), DRM boosted ARC2-Eval performance from 6.3%[3] to 9.6% and the performance of TRM without deep supervision was just 3.3%. Even in the high-data regime (NVARC pretraining), DRM improved ARC2-Eval accuracy from 12.5% to 16.7% (7M params) and 21.8% to 24.9% (14M params).

Performance Scaled with Data and Model Size. We observe a strong synergistic effect between model and dataset size. Scaling from the 7 million parameter baseline (and four recurrent gradients steps) to the 14 million parameter (and six recurrent gradients steps) improved performance 3–4% when restricted to pretraining on ARC2-Train. The increased capacity from a bigger model became more critical when leveraging massive pretraining datasets. Adding NVARC to pretraining improved 7M DRM by approximately 7% but with 14 million parameters, this boosted performance by 11.5%. This indicates that the smaller model is capacity constrained, whereas the 14M architecture can absorb the high diversity of the 47K tasks from NVARC.

DRM Outperformed SOTA 4B LLM Method. Our 14M-parameter DRM outperformed the current state-of-the-art NVARC (Qwen3-4B Base) model (24.9% vs 22.6%) when trained on the same data. This result is particularly striking because NVARC is a highly engineered system built around a 4-billion parameter model, specialized specifically for ARC-AGI. This demonstrated that DRM (and TRM) can be orders of magnitude more parameter efficient (14M vs. 4B).

State Perturbation Recursion Model. Our SPRM method also improved over the TRM baseline (50.7% vs. 45.7% on ARC2-Train and 6.3% vs. 10.1% on ARC2-Eval). However, its effectiveness was mixed in other settings. On the reARC dataset, SPRM performed slightly worse than the TRM baseline. We hypothesize that the high diversity of reARC (1000 examples per task) saturated the model's need for exploration, making noise injection redundant.

[3]Jolicoeur-Martineau (2025) reported 7.8% in their paper with the identical setup.

6 DISCUSSION

ARC vs. other settings where diffusion excels. Our results highlight the particular type of iterative refinement needed for ARC-style algorithmic reasoning. Discrete masked diffusion baselines fail almost completely even when scaled to match the TRM compute budget. ARC is a discriminative point-to-point mapping rather than to a distribution-to-distribution generative setting where diffusion usually excels. In image generation, denoising can often progress from coarse global structure to fine local detail, and many outputs may be acceptable. ARC has a single exact target, and many tasks require globally coordinated discrete transformations where locally plausible partial states are incompatible with the final solution. This mismatch may help explain why standard masked diffusion performs poorly even though diffusion is highly effective in conventional generative domains.

Importance of Backpropagating Through Recursion. While DRMs substantially improve over TRMs, the difference in performance is much more modest than the gap between TRM and diffusion baselines. These findings suggest that the core ingredient to the success of these small models is backpropagating through a window of recursions. We see two complementary mechanisms. First, this provides *multi-step credit assignment*: intermediate states are trained to be useful because they enable progress several steps later, which discourages greedy updates that could be very poor for certain tasks—e.g., in obstacle navigation, a move that decreases Manhattan distance can still result in a dead end. Recursive denoising creates conditions under which planning-like latent computation (Guez et al., 2019) could emerge. Second, weight tying across depth can act as a strong regularizer for algorithmic reasoning. Because the same parameters must operate across many latent states and depths, gradients from different unrolled steps must largely agree for an update to persist; spurious, step-specific correlations are more likely to generate conflicting signals that cancel, biasing learning toward stable, reusable computations.

DRM balances "off-policy" and "on-policy" learning. DRM improves upon TRM in part by mitigating long-horizon training instability. Pure backward-training is effectively "on-policy": the model must discover long trajectories from noise using only its own self-generated intermediate states, which are often uninformative early in training. DRM balances "off-policy" and "on-policy" learning. It initializes training from diffusion-corrupted targets, providing an "off-policy" curriculum of easier intermediate states closer to the solution. From that starting point onward, training is explicitly "on-policy": the model is unrolled for multiple recursive denoising steps, and each later state is generated by the model itself rather than supplied by the corruption process. This preserves the forward-looking behavior and train–test alignment that make TRM effective.

Scaling Behaviour. Finally, we provide evidence of scaling in these TRM-based recursion models. The strong interaction between model size, unrolled gradient depth, and pretraining scale suggest that recursion models are not limited to the "tiny" regime, rather their parameter efficiency can be converted into improved performance as data and compute increase.

7 CONCLUSIONS

We introduced the Denoising Recursion Model (DRM), a looped transformer that resolves the opposing limitations of (1) backward-training methods that struggle with long-horizon stability and (2) diffusion that suffers from a myopic objective and a misalignment between training (on noised target) and testing (on self-generated trajectories). We effectively combine the training objectives: we corrupt the target with noise identical to the diffusion process, but train the model to recover the clean signal over a sequence of recursive steps, rather than a single jump. Our method consistently outperformed the TRM baseline on ARC-AGI across a variety of settings. On the largest pretraining dataset, we achieved 24.9% accuracy on ARC2-Eval, surpassing the winner of the recent ARC Challenge (NVARC) when trained on identical data. We also showed large improvements from scaling the architecture from 7M to 14M parameters, suggesting that further scaling could yield substantial gains. At present, progress on ARC-AGI is difficult to interpret because gains from model architecture and training objectives are often entangled with gains from more sophisticated synthetic-data generation. Our results suggest that substantial headroom remains in the learning algorithm itself, and motivate evaluations that make these sources of progress easier to disentangle.

REFERENCES

Akyürek, E., Damani, M., Zweiger, A., Qiu, L., Guo, H., Pari, J., Kim, Y., and Andreas, J. (2024). The surprising effectiveness of test-time training for few-shot learning. arXiv preprint arXiv:2411.07279.

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and Van Den Berg, R. (2021). Structured denoising diffusion models in discrete state-spaces. Advances in neural information processing systems, 34:17981–17993.

Bai, S., Kolter, J. Z., and Koltun, V. (2019). Deep equilibrium models. Advances in neural information processing systems, 32.

Bai, S., Koltun, V., and Kolter, J. Z. (2021). Stabilizing equilibrium models by jacobian regularization. arXiv preprint arXiv:2106.14342.

Bengio, E., Jain, M., Korablyov, M., Precup, D., and Bengio, Y. (2021). Flow network based generative models for non-iterative diverse candidate generation. Advances in neural information processing systems, 34:27381–27394.

Black, K., Janner, M., Du, Y., Kostrikov, I., and Levine, S. (2024). Training diffusion models with reinforcement learning. In The Twelfth International Conference on Learning Representations.

Bush, T., Chung, S., Anwar, U., Garriga-Alonso, A., and Krueger, D. (2025). Interpreting emergent planning in model-free reinforcement learning. In The Thirteenth International Conference on Learning Representations.

Chang, H., Zhang, H., Jiang, L., Liu, C., and Freeman, W. T. (2022). Maskgit: Masked generative image transformer. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 11315–11325.

Chen, B., Martí Monsó, D., Du, Y., Simchowitz, M., Tedrake, R., and Sitzmann, V. (2024). Diffusion forcing: Next-token prediction meets full-sequence diffusion. Advances in Neural Information Processing Systems, 37:24081–24125.

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

Chollet, F., Knoop, M., Kamradt, G., Landers, B., and Pinkard, H. (2025). Arc-agi-2: A new challenge for frontier ai reasoning systems. arXiv preprint arXiv:2505.11831.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. (2015). A recurrent latent variable model for sequential data. Advances in neural information processing systems, 28.

Cole, J. and Osman, M. (2025). Don't throw the baby out with the bathwater: How and why deep learning for arc. arXiv preprint arXiv:2506.14276.

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

Dieng, A. B., Ranganath, R., Altosaar, J., and Blei, D. (2018). Noisin: Unbiased regularization for recurrent neural networks. In International Conference on Machine Learning, pages 1252–1261. PMLR.

Elman, J. L. (1990). Finding structure in time. Cognitive science, 14(2):179–211.

Fan, Y., Du, Y., Ramchandran, K., and Lee, K. (2025). Looped transformers for length generalization.

Franzen, D., Disselhoff, J., and Hartmann, D. (2025a). The architects arc prize 2025 solution: Recursive masked diffusion for 2d reasoning. ARC Prize 2025 Technical Reports.

Franzen, D., Disselhoff, J., and Hartmann, D. (2025b). Product of experts with llms: Boosting performance on arc is a matter of perspective. arXiv preprint arXiv:2505.07859.

Gao, Z., Chen, L., Xiao, Y., Xing, H., Tao, R., Luo, H., Zhou, J., and Dai, B. (2025). Universal reasoning model. arXiv preprint arXiv:2512.14693.

Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., and Goldstein, T. (2025a). Scaling up test-time compute with latent reasoning: A recurrent depth approach. arXiv preprint arXiv:2502.05171.

Geiping, J., Yang, X., and Su, G. (2025b). Efficient parallel samplers for recurrent-depth models and their connection to diffusion language models. arXiv preprint arXiv:2510.14961.

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

Guez, A., Mirza, M., Gregor, K., Kabra, R., Racanière, S., Weber, T., Raposo, D., Santoro, A., Orseau, L., Eccles, T., et al. (2019). An investigation of model-free planning. In International conference on machine learning, pages 2464–2473. PMLR.

He, H., Renz, K., Cao, Y., and Geiger, A. (2025). Mdpo: Overcoming the training-inference divide of masked diffusion language models. arXiv preprint arXiv:2508.13148.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in neural information processing systems, 33:6840–6851.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8):1735–1780.

Hodel, M. (2024). Addressing the abstraction and reasoning corpus via procedural example generation. arXiv preprint arXiv:2404.07353.

Hu, K., Cy, A., Qiu, L., Ding, X. D., Wang, R., Zhu, Y. E., Andreas, J., and He, K. (2025). Arc is a vision problem! arXiv preprint arXiv:2511.14761.

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

Lai, C.-H., Song, Y., Kim, D., Mitsufuji, Y., and Ermon, S. (2025). The principles of diffusion models. arXiv preprint arXiv:2510.21890.

Lee, J., Mansimov, E., and Cho, K. (2018). Deterministic non-autoregressive neural sequence modeling by iterative refinement. arXiv preprint arXiv:1802.06901.

Lim, S. H., Erichson, N. B., Hodgkinson, L., and Mahoney, M. W. (2021). Noisy recurrent neural networks. Advances in Neural Information Processing Systems, 34:5124–5137.

Merrill, W. and Sabharwal, A. (2025). A little depth goes a long way: The expressive power of log-depth transformers. arXiv preprint arXiv:2503.03961.

Moskvichev, A., Odouard, V. V., and Mitchell, M. (2023). The conceptarc benchmark: Evaluating understanding and generalization in the arc domain.

Nichol, A. Q. and Dhariwal, P. (2021). Improved denoising diffusion probabilistic models. In International conference on machine learning, pages 8162–8171. PMLR.

Nie, S., Zhu, F., You, Z., Zhang, X., Ou, J., Hu, J., ZHOU, J., Lin, Y., Wen, J.-R., and Li, C. (2025). Large language diffusion models. In The Thirty-ninth Annual Conference on Neural Information Processing Systems.

OpenAI (2025). Introducing OpenAI o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/.

Poetiq Team (2025). Traversing the frontier of superintelligence. Poetiq AI Blog. Accessed: 2026-01-26.

Pourcel, J., Colas, C., and Oudeyer, P.-Y. (2025). Self-improving language models for evolutionary program synthesis: A case study on arc-agi. arXiv preprint arXiv:2507.14172.

Saunshi, N., Dikkala, N., Li, Z., Kumar, S., and Reddi, S. J. (2025). Reasoning with latent thoughts: On the power of looped transformers. arXiv preprint arXiv:2502.17416.

Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. (2018). Learning a sat solver from single-bit supervision. arXiv preprint arXiv:1802.03685.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In International conference on machine learning, pages 2256–2265. PMLR.

Sorokin, I. and Puget, J.-F. (2025). NVARC solution to ARC-AGI-2 2025. Kaggle ARC Prize 2025, 1st Place Solution. Also available at `https://github.com/1ytic/NVARC`.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. (2024). Roformer: Enhanced transformer with rotary position embedding. Neurocomputing, 568:127063.

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

Williams, R. J. and Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. In Backpropagation: theory, architectures, and applications, pages 433–486.

Xu, Y., Khalil, E. B., and Sanner, S. (2023). Graphs, constraints, and search for the abstraction and reasoning corpus. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, pages 4115–4122.

You, Z., Nie, S., Zhang, X., Hu, J., Zhou, J., Lu, Z., Wen, J.-R., and Li, C. (2025). Llada-v: Large language diffusion models with visual instruction tuning. arXiv preprint arXiv:2505.16933.

Zhou, C., Yang, C., Hu, Y., Wang, C., Zhang, C., Zhang, M., Mackey, L., Jaakkola, T., Bates, S., and Zhang, D. (2025). Coevolutionary continuous discrete diffusion: Make your diffusion language model a latent reasoner.

Zhu, F., Wang, R., Nie, S., Zhang, X., Wu, C., Hu, J., Zhou, J., Chen, J., Lin, Y., Wen, J.-R., et al. (2025a). Llada 1.5: Variance-reduced preference optimization for large language diffusion models. arXiv preprint arXiv:2505.19223.

Zhu, R.-J., Wang, Z., Hua, K., Zhang, T., Li, Z., Que, H., Wei, B., Wen, Z., Yin, F., Xing, H., Li, L., Shi, J., Ma, K., Li, S., Kergan, T., Smith, A., Qu, X., Hui, M., Wu, B., Min, Q., Huang, H., Zhou, X., Ye, W., Liu, J., Yang, J., Shi, Y., Lin, C., Zhao, E., Cai, T., Zhang, G., Huang, W., Bengio, Y., and Eshraghian, J. (2025b). Scaling latent reasoning via looped language models.

A Background

A.1 ARC-AGI

While modern AI systems excel at tasks where massive training data is available, they struggle with test-time adaptation—the ability to acquire new skills on the fly during deployment. Accurately evaluating this adaptability is difficult: given the scale of modern pretraining corpora, it is nearly impossible to verify whether a benchmark task is truly novel to a model or simply recalled from memory. To address this challenge, Chollet (2019) introduced the Abstraction and Reasoning Corpus (ARC-AGI): a diverse set of novel tasks where each task provides a few-shot context (2–5 input-output grid pairs). See Figure 2 for an example.

Each task is defined by a singular, simplest latent algorithm that uniquely explains the transformation across the examples. Solving a task requires reverse-engineering this algorithm to determine the exact output grid for an unseen test input. Since each task is unique and shares only abstract "core knowledge priors" (e.g., objectness, symmetry), models cannot rely on retrieving pre-learned

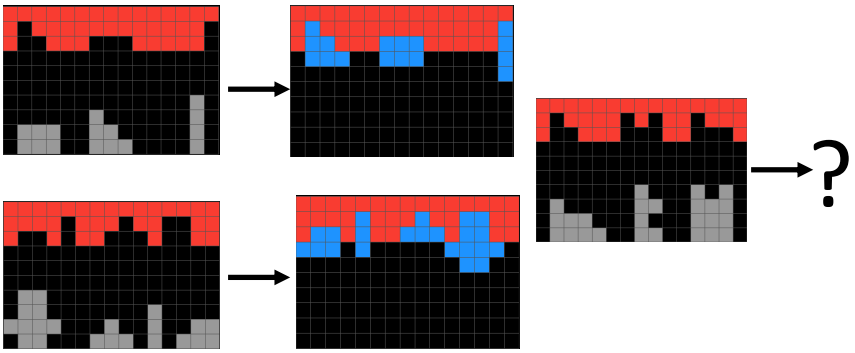

Figure 2: Example of ARC-AGI task. Two input-output grids examples for the task transformation: move each gray blocks in the input into the corresponding slots in the output and change colours to blue. Must apply the transformation correctly to a held-out input to be correct.

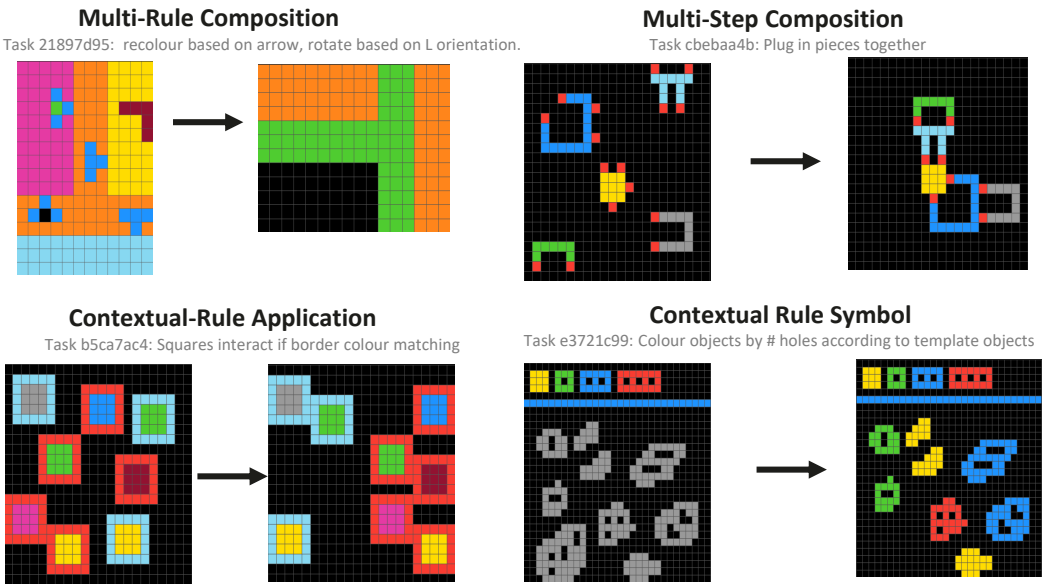

Figure 3: Various examples of the flavours of reasoning needed to solved ARC-AGI-2 tasks.

solutions. See Figure 3 for diverse examples of ARC-AGI-2 tasks. Instead, they must perform explicit new learning to induce the transformation rule. ARC-AGI remains a formidable challenge for AI, serving as a primary leaderboard for progress toward general intelligence (Chollet et al., 2025).

While some early work attempted to solve ARC-AGI using discrete search (Xu et al., 2023), neural methods now dominate the literature. These approaches can be broadly categorized into *program synthesis* and *direct grid prediction*. A program synthesis approach is currently state of the art on ARC-AGI-2. Poetiq Team (2025) leveraged top-tier closed-source LLMs to generate code solutions mapping input to output grid and use iterative prompting strategies to verify and refine solutions. Code generation tends to be not nearly as advantageous with open-source models, though Pourcel et al. (2025) recently achieved competitive results by constructing a large dataset reinterpreting its self-generated incorrect programs as correct programs for new tasks.

Our work primarily compares to open-source *direct grid prediction* methods, which treat ARC as a token or image generation problem. The standard recipe for these models involves three components: (1) pretraining on large-scale synthetic datasets, (2) test-time fine-tuning on augmentations of the few-shot examples, and (3) search-based decoding with selection mechanisms like majority voting (Cole and Osman, 2025; Akyürek et al., 2024; Franzen et al., 2025b). The winner of the 2025 ARC Challenge (Chollet et al., 2025), NVARC (Sorokin and Puget, 2025), followed this formula using

Franzen et al. (2025b)'s approach, with the critical innovation being a clever mechanism to leverage LLMs to generate a massive, diverse synthetic dataset for pretraining.

The second-place solution (Franzen et al., 2025a) is particularly relevant as it adapts Llada (Nie et al., 2025), a pre-trained diffusion language model. However, their approach differs fundamentally from ours. They rely on a standard, fixed-depth transformer architecture with 4 billion parameters and repeatedly re-masking and denoising at test time. In contrast, our method uses diffusion to train a recursive, parameter-efficient loop from scratch.

Beyond architecture, recursion models differ from these LLM-based approaches in how they process few-shot examples. LLM approaches rely on *in-context learning*, where all input-output pairs are concatenated into a single prompt. In contrast, HRM, TRM, and URM all train a shared *task embedding* with the few-shot examples. The model then conditions on this embedding to process the test input. Conceptually, the forward pass in an LLM must simultaneously infer the transformation and apply it; in these models, the forward pass can be viewed as "executing" the learned task program (the embedding) on the new input.

B DRM REASONNG EXAMPLES

We now show a few examples of DRM execution on test examples from ARC-AGI-2 Eval. During inference, DRM takes 16 denoising steps. We show the sequential predictions plus the remaskings for each step according to our noise schedule. We show the input and ground truth label at top of the image and the internal confidence $\hat{P}$(correct) across steps in the top right. Each prediction grid is annotated with the internal confidence and each remasking grid is annotated with remasking ratio—the fraction of cells being remasked.

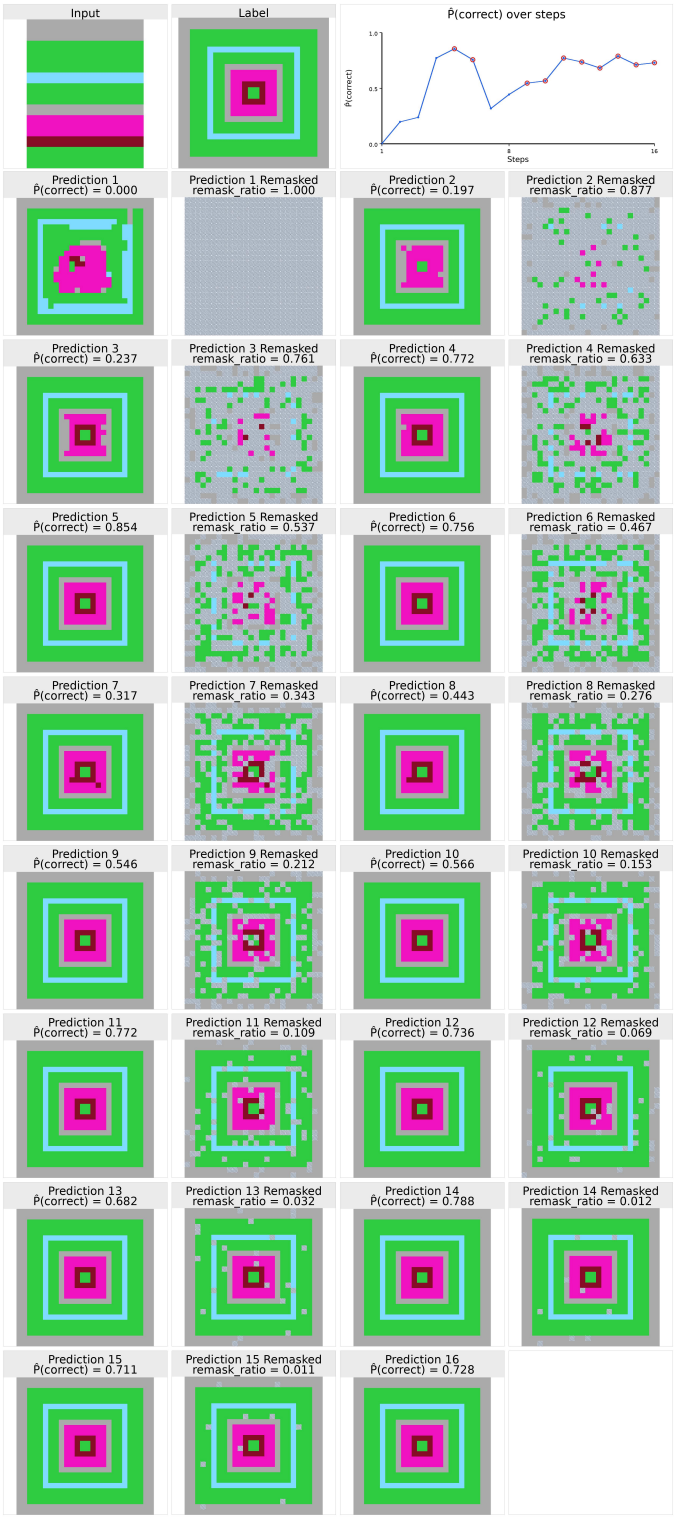

Figure 4: Successful convergence on the ARC-AGI 2 problem of constructing nested squares with a certain colour order specified in the input. The model initially generates an incorrect square with a rough approximation of the correct answer and it refines it over several steps and eventually converges to the correct answer. Self-confidence $\hat{P}$(correct) is highly correlated with correctness indicating the model understood when it was correct and incorrect.

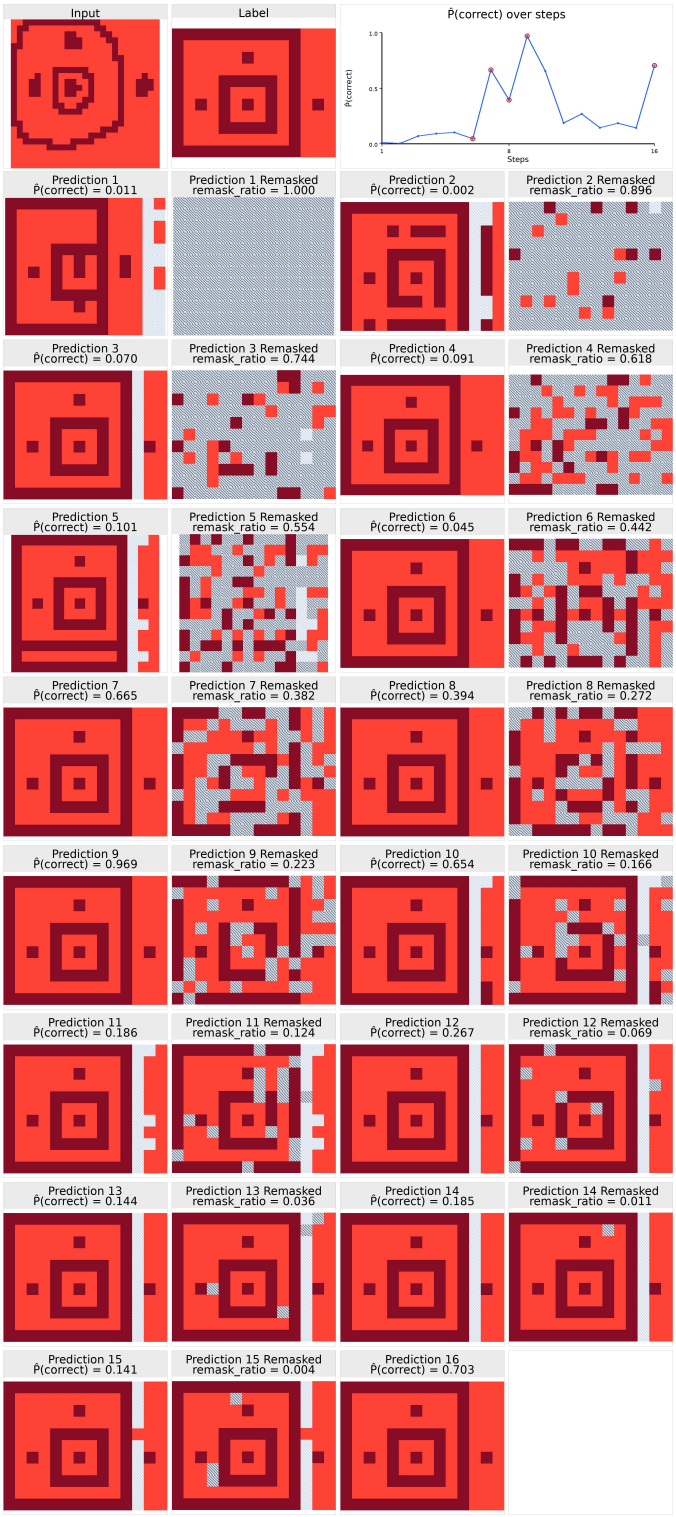

Figure 5: Successful solution on the ARC-AGI 2 problem of identifying core shapes from a rough "sketch". Model is initially confused but eventually makes progress before oscillating between correct and incorrect solution and happens to settle on the correct answer in the final step.

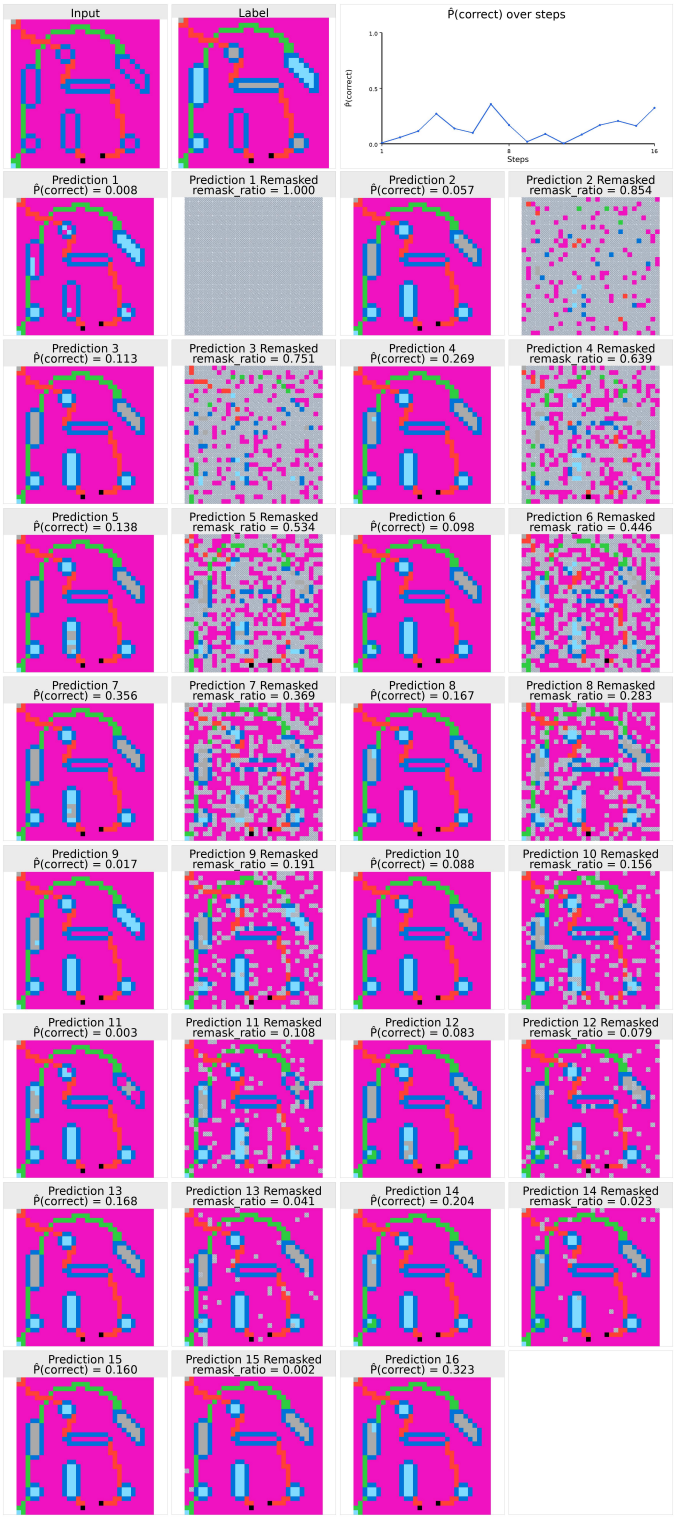

Figure 6: Failure to converge to a correct answer on a ARC-AGI 2 task to fill in shapes corresponding to what colour path they lie on. The model seems to understand that the internals of the shapes must be filled but it clearly does not know what the colouring rule is. However, the model shows very low internal confidence indicating it knows it is confused.

C PSEUDOCODE

See Figure 7 for pseudocode comparing the training logic between TRM and DRM.

(a) TRM (Deep Supervision)

```
def trm_rec(x, y, z, n=6, T=3):
   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, z), dec(y), Q(y)

for x, y_true in train_data:
   # random initialization
   y, z = y_rand, z_rand
   # iter over recursion windows
   for step in N:
     x = input_embed(x)
     (y,z),y_hat,q_hat =
trm_rec(x,y,z)
     loss = ce(y_hat, y_true)
     loss += bce(q_hat,
(y_hat==y_true))
     loss.backward()
     opt.step()
     # truncated backpropagration
     opt.zero_grad()
     # early-stopping
     if q_hat > 0:
       break
```

(b) DRM (Ours)

```
def drm_rec(x, y, z, n=6, T=3):
   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, z), dec(y), Q(y)

for x, y_true in train_data:
   z = z_rand
   # noise target
   tau = sample_time_step()
   yn = add_noise(y_true, tau)
   yn = label_embedding(yn)
   x = input_embedding(x)
   (y,z),y_hat,q_hat =
drm_rec(x,yn,z)
   loss = ce(y_pred, y_true)
   loss += bce(q_hat,
(y_hat==y_true))
   loss.backward()
   opt.step()
```

Figure 7: Comparison of training logic between TRM and DRM (with TRM base). Text in red highlights logic that only appears in TRM, text in green highlights logic that only appears in DRM. (a) TRM uses truncated backpropagation. (b) Denoising Recursion uses a noise curriculum without truncated backpropagation.

D TRAINING DETAILS

See Table 2 for training details and Table 3 for archicture details.

E EXTENDED RESULTS

E.1 PASS@K RESULTS

Figure 8 shows pass@k ARC2-Eval results for the 14M evaluations shown in Table 1. When pretrained on the largest NVARC dataset, DRM shows a clear improvement up to pass@1000, where it is $\sim 5\%$ better than the TRM baseline. However, when pretraining on ARC2-train, TRM matches DRM performance at pass@100 and pass@1000. The gap between pass@1 and pass@1000 shows the potential for further improvement. We believe that generation of the correct output is much more difficult than identifying the correct output over a candidate set—i.e., we hypothesize humans would easily discriminate over correct and incorrect outputs.

Hyperparameter	Train-from-scratch	Pretrain		Finetune
		reARC	NVARC	
Epochs	100k	100k	2k	300k
LR	1e-4	1e-4	1e-4	1e-4
Puzzle_emb_lr	1e-2	1e-2	1e-2	1e-2
lr_warmup_steps	2k	10k	2k	2k
lr_anneal	no	no	no	no
Batch size	768	768	768	768
Weight decay	0.1	0.1	0.1	0.1
EMA	True	True	True	True
Loss	CE	CE	CE	CE
Optimizer	AdamW	AdamW	AdamW	AdamW

Table 2: TRM training hyperparameters for each stage.

Hyperparameter	TRM/SPRM/DRM		TRM (loop untie)	Transformer
	7M	14M		
Layers	2	2	2	10
Warm-up Windows (H cycles)	3	3	3	1
Gradient Loops (L cycles)	4	6	4(untied)	1
Hidden size	512	768	512	768
Num heads	8	12	8	12
Expansion	4	4	4	4
Max halt steps	16	16	16	16

Table 3: Model/config hyperparameters across methods.

E.2 Missing Baselines

Due to computational constraints, we do not benchmark the second-place solution (finetuned from Llada diffusion language model) (Franzen et al., 2025a). We attempted to reproduce ViARC (Hu et al., 2025) and the contemporaneous URM (Universal Reasoning Model) (Gao et al., 2025) but were unable to get stable results. ViARC showed severe optimization instability and URM suffered from persistent training crashes.

E.3 TRM Extreme Robustness to Overfitting

Figure 9 illustrates an unusual training dynamic: validation continues to improve well after the model has effectively "solved" the training set. In particular, once training exceeds 99% per-pixel exact accuracy across the entire output grid (100k steps), validation metrics keep rising from 38% to 45% between 100k and 500k steps. Note that here we are training on ARC2-train with $\sim$3k examples. Since our batch size is 768, four steps is virtually one epoch. This means that TRM does not overfit even after seeing a example about 100k times! Note that we do rotate through 1000 different "augmentations" per data point (e.g., colour permutation, rotation), which does some of the work to avoid overfitting to basic artifacts.

When we instead stack layers without looping, the generalization is much worse. Figure 10 shows that while training loss converges much faster (left) when we stack (Transformer), exact-match training performance of the looped model (TRM) quickly surpasses the stacked model (centre). This suggests that "global" understanding is clearly worse when stacking. This translates to much better validation performance (right).

E.4 SPRM is a good regularization of input

SPRM injects noise into the latent state that is carried over and used as the input to the next refinement step. We view noise injection as a regularizer, promoting generalization by increasing the diversity of

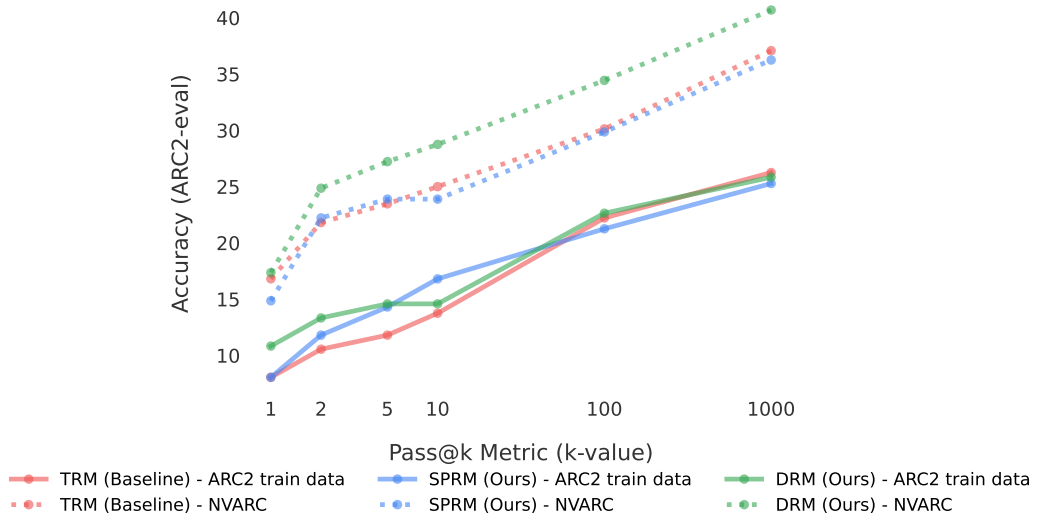

Figure 8: Pass@k exact-match accuracy on ARC2-Eval for three methods (TRM baseline, SPRM, and DRM) with 14M parameters and pretrained with two scales of data (ARC2-train and NVARC)

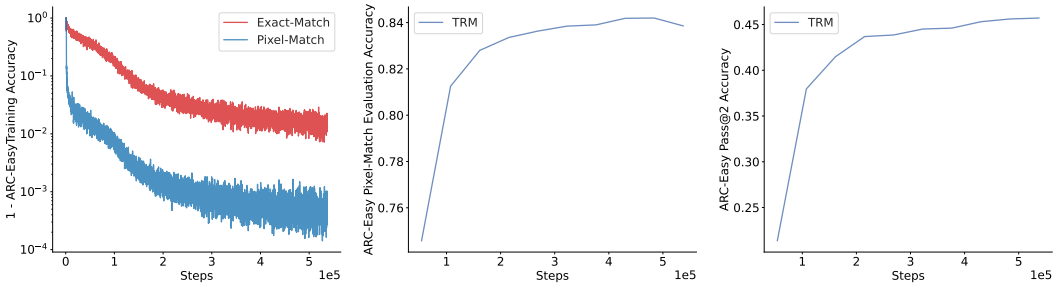

Figure 9: **Extreme robustness to overfitting.** TRM validation accuracy continues to improve even after training surpasses 99% per-pixel accuracy across the grid. (left) 1–exact-match accuracy, 1–per-cell accuracy, (middle) validation per-cell accuracy, (right) validation pass@2.

training states because the same underlying example can generate many nearby latent inputs across refinement steps.

Figure 11 further indicates that SPRM outperform TRM from an optimization perspective. Adding noise actually improves convergence during training. By "thickening" the refinement trajectory, it may smooth the effective loss landscape since gradients are averaged over latent perturbations rather than tied to one precise hidden trajectory. It is counterintuitive how adding noise could improve training loss and accuracy but this does align with prior findings on RNNs that injecting noise into intermediate recurrent states can promote a contractive, stable recursion (Lim et al., 2021). We also note that noising intermediate steps is similar to Variational RNNs (Chung et al., 2015) which introduce latent random variables into the recurrence. Prior work on RNNs showed that noise injection can outperform Dropout (Dieng et al., 2018).

E.5 Latent "scratchpad" is not needed

HRM (Wang et al., 2025) maintains two latent states: a low-level state z_L that repeatedly processes the input signal, and a high-level state z_H that acts as a slower, summarizing carrier across cycles. z_L is essentially a scratchpad state that continually is carried forward while z_H is decoded into a solution. TRM adopts a similar two-latent-state design but simplifies the architecture by tying the high-level update to the same network used for low-level transitions. The HRM pattern:

$$z_L \leftarrow f_L(z_L,\, z_H + x), \quad z_H \leftarrow f_H(z_H,\, z_L)$$

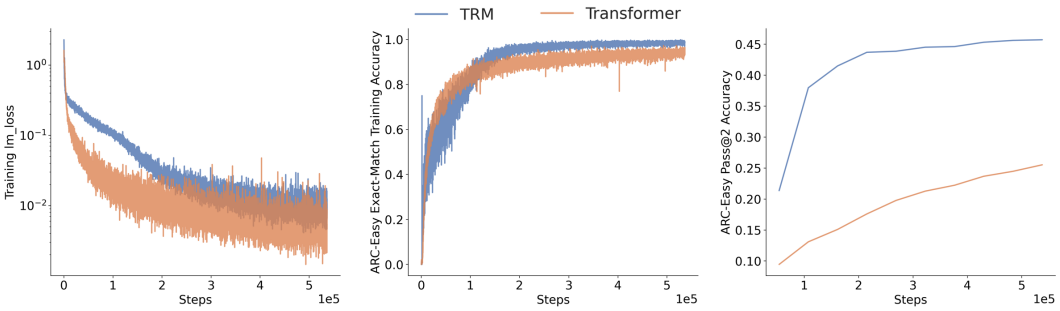

Figure 10: Unlooped transformer has better training loss than TRM baseline (left)) but worse training exact-match accuracy (centre) and higher evaluation pass@2 accuracy (right).

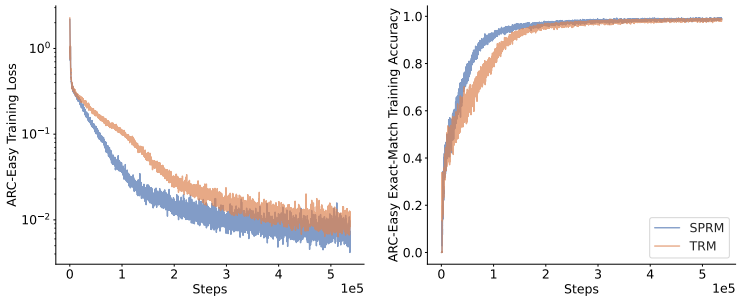

Figure 11: Training dynamics comparison between TRM and SPRM. (left) training exact accuracy and (right) training loss.

becomes an update where both transitions reuse a single module f:

$$z_L \leftarrow f(z_L,\, z_H + x), \quad z_H \leftarrow f(z_H,\, z_L).$$

A natural question is whether this scratchpad z_L is truly necessary. In Jolicoeur-Martineau (2025)'s ablations, they test a more aggressive simplification that only depends on z_H

$$z_H \leftarrow f(z_H, x), \qquad z_H \leftarrow f(z_H).$$

which performed substantially worse. In contrast, our ablation shows that the scratchpad z_L is not needed if we get rid of this "summarization" step:

$$z_H^{(t+1)} \leftarrow f\left(z_H^{(t)},\, x\right).$$

Pass@2 exact-match accuracy remains on par with the baseline as shown in Figure 12 on ARC2-Easy. This suggests that the performance drop observed in their ablation is not due to removing z_L, but rather due to introducing an extra latent-only transition $z_H \leftarrow f_H(z_H)$.

This simplification was also made by the contemporaneous URM (Gao et al., 2025) and we argue this is cleaner since the recursion becomes perfectly homogeneous across iterations.

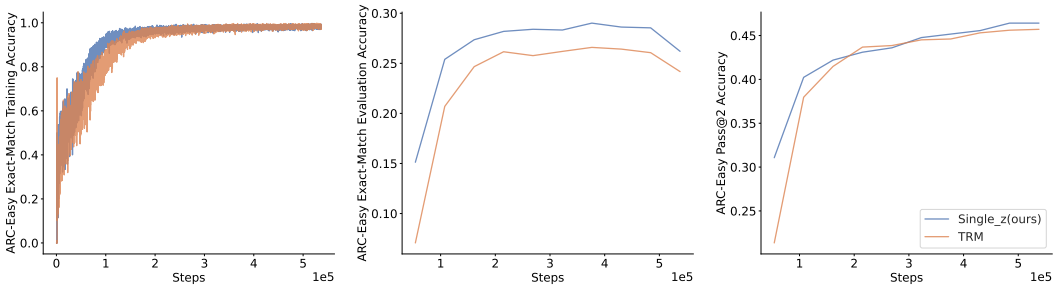

Figure 12: Single_z (no scratchpad latent state) performed as good as the TRM baseline.