LEARNING TO REFINE PARTIAL SOLUTIONS WITH RECURSIVE DISCRETE DENOISING

Anonymous authors
Paper under double-blind review

ABSTRACT

Structured reasoning requires predictions whose parts jointly satisfy constraints or an inferred rule. Reusing a small neural network across computation steps offers a parameter-efficient approach, but each intermediate answer can contain both useful assignments and errors. Learning to refine such answers is challenging when supervision provides completed solutions without intermediate reasoning trajectories. We propose the Discrete Reasoning Diffusion Model (DRDM), which trains a looped Transformer to reconstruct solutions from partially correct candidates. Masking and token replacement create completion and correction tasks while preserving the observed problem. A shared core performs several latent updates before predicting solution tokens; progressive decoding adds assignments, allows revision, and carries latent memory between steps. Across three training seeds, the approximately 7M-parameter model achieves 85.2% exact accuracy on Sudoku-Extreme, 87.3% on Maze-Hard, and 50.5% pass@2 on ARC-AGI-1, exceeding self-attention TRM by 9.7, 2.7, and 7.3 percentage points, respectively.

1 INTRODUCTION

Structured reasoning requires predictions that satisfy relations across an entire solution. A Sudoku entry must agree with its row, column, and box; a maze path must connect distant endpoints; an ARC output must apply a rule inferred from demonstrations. Local predictions can constrain later decisions, but a plausible assignment can also create conflicts elsewhere. Solving these problems through repeated updates gives a learned model opportunities to propagate information and reconsider its intermediate answer.

Recurrent architectures provide a compact way to perform this computation. A *looped Transformer* repeatedly applies shared Transformer layers to an evolving representation, increasing computational depth through reuse (Dehghani et al., 2019). Recurrent networks have been studied for generalization to harder reasoning problems (Schwarzschild et al., 2021), and recurrent-depth language models extend this approach to additional latent computation at inference time (Geiping et al., 2025). For structured puzzles, the Hierarchical Reasoning Model (HRM) uses two recurrent modules operating at different frequencies (Wang et al., 2025a). The Tiny Recursive Model (TRM) simplifies this organization to a small shared network that repeatedly updates a latent state and an answer representation (Jolicoeur-Martineau, 2025). These results make learned recurrence a compelling setting for parameter-efficient reasoning.

An important training question remains: *which intermediate solutions should a recurrent solver learn to refine?* An incomplete answer may contain both useful assignments and mistakes. A correct Sudoku digit narrows the remaining possibilities, whereas an incorrect digit can mislead subsequent predictions. Supervision at successive recurrent steps supplies the desired answer, but the candidate supplied at each step determines the refinement problem the network encounters. In the setting considered here, training provides problem–solution pairs, without annotated sequences of intermediate decisions. We seek a training formulation that exposes a small recurrent solver to candidates with different amounts of correct, missing, and incorrect information.

Denoising offers a way to construct these learning problems from completed solutions. Reconstruction from corrupted observations underlies denoising autoencoders and learned generative transitions (Vincent et al., 2008; Bengio et al., 2014); it also motivates conditional iterative sequence refinement (Lee et al., 2018). Discrete diffusion makes the corruption distribution an explicit design choice

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

(Austin et al., 2021; Sahoo et al., 2024). For a structured solution, masking creates missing assignments, while token replacement creates assignments that must be corrected. Both can be sampled without changing the observed problem or requiring an intermediate-state annotation. Figure 2 illustrates how this construction turns a completed solution into distinct completion and correction tasks.

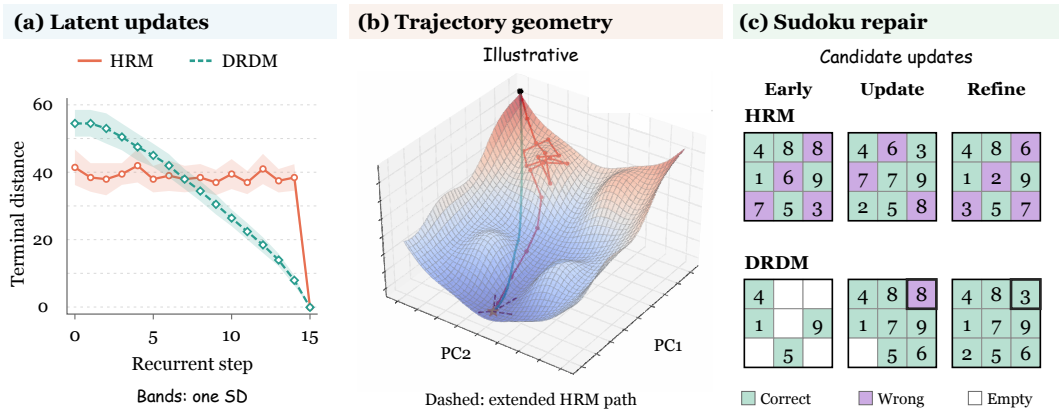


Figure 1: **Three views of recursive refinement.** (a) HRM and DRDM distances to their terminal hidden states; bands show one SD. (b) Illustrative trajectory geometry. (c) Constructed candidate sequences contrast changing incorrect assignments with progressive filling and correction. Recorded Sudoku predictions appear in Appendix Figure 13.

We introduce the Discrete Reasoning Diffusion Model (DRDM), a recursive denoiser trained to complete and correct these candidates. Within each denoising step, a shared Transformer core processes the candidate and task context through several latent updates. Confidence-based decoding then updates the visible assignments, while latent memory carries information into the next step.

Across three training seeds, the approximately 7M-parameter DRDM improves over self-attention TRM by 9.7 percentage points on Sudoku-Extreme, 2.7 on Maze-Hard, and 7.3 on ARC-AGI-1. We examine how reconstruction training, consistency regularization, and the decoding horizon affect this performance. Figure 1 connects the latent updates to visible candidate assignments; Section 4.6 examines the corresponding trajectory diagnostics.

2 Background and Related Work

2.1 Recurrent Computation for Reasoning

Shared recurrence allows a model to spend more computation without adding parameters. With finite unrolling, the number of updates can be learned (Graves, 2016), and self-attention can be reused across recurrent depth (Dehghani et al., 2019). Looped Transformers can thereby learn iterative algorithms, as demonstrated in data-fitting tasks (Yang et al., 2024). Equilibrium formulations seek a self-consistent representation at a fixed point of the shared transformation (Bai et al., 2019). Equilibrium Reasoners learn task-conditioned attractors (Huang et al., 2026), while Fixed-Point Reasoners study stable recurrent dynamics and adaptive computation (Movahedi et al., 2026).

For reasoning tasks, the usefulness of additional steps depends on the update learned during training. Neural algorithmic reasoning studies reusable computations over structured states (Veličković & Blundell, 2021), with repeated relational message passing providing a concrete example in Sudoku (Palm et al., 2018). Applying a trained update to problems requiring longer computations raises the question of how recurrence generalizes (Schwarzschild et al., 2021). Progressive training and input recall help sustain useful recurrence (Bansal et al., 2022), while recurrent-depth language models explore additional latent computation at inference time (Geiping et al., 2025). In the compact solvers HRM and TRM, intermediate predictions are supervised against completed solutions (Wang et al., 2025a; Jolicoeur-Martineau, 2025). This leaves a further training choice: which partial answers the solver should encounter, including assignments that are informative but potentially incorrect.

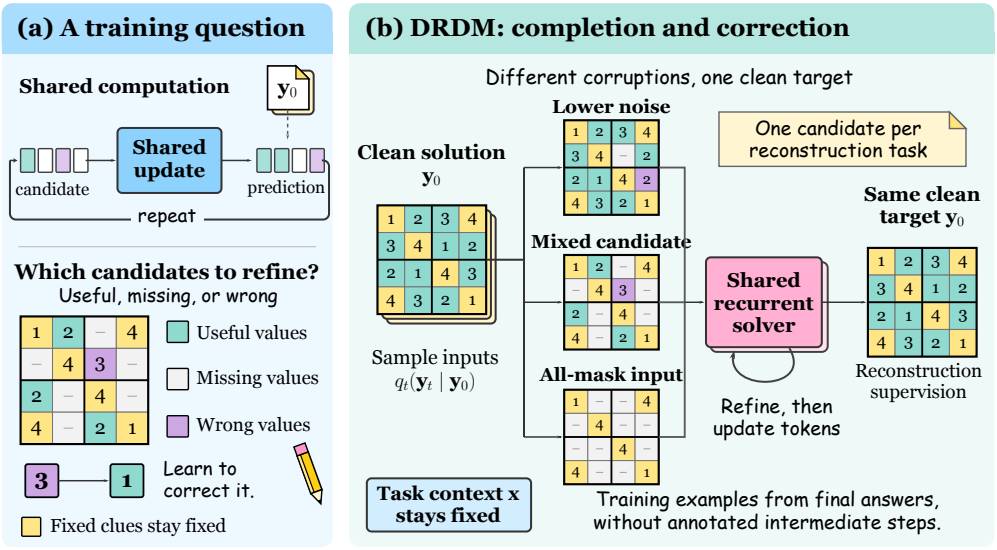


Figure 2: **Learning to complete and correct partial solutions.** (a) Recurrence leaves the training candidates unspecified. (b) Independent corruptions retain the same clean target: fixed clues (yellow), retained values (mint), masks (gray), and replacements (purple).

2.2 Denoising as a Refinement Learning Problem

Denoising provides a way to construct such training inputs from completed examples. Reconstruction of corrupted observations serves both representation learning in denoising autoencoders (Vincent et al., 2008) and generative modeling through learned reconstruction transitions, including transitions with latent state (Bengio et al., 2013; 2014). For conditional prediction, the same idea makes a provisional answer an input to a subsequent refinement step (Lee et al., 2018). Mask-Predict gives a concrete instance: training uses partially observed targets, and decoding revisits low-confidence tokens (Ghazvininejad et al., 2019). The training distribution therefore determines what kinds of incomplete answers the model learns to use.

Diffusion extends reconstruction across a range of noise levels (Sohl-Dickstein et al., 2015; Ho et al., 2020). Its design separates the training noise distribution, predictor, and sampling procedure (Karras et al., 2022). For categorical data, the corruption kernel controls which values are retained and which are replaced or masked (Hoogeboom et al., 2021; Austin et al., 2021); the transitions can also be formulated as a continuous-time Markov jump process (Campbell et al., 2022). The resulting denoising problem admits objectives based on probability ratios (Lou et al., 2024) or weighted clean-token cross-entropy (Shi et al., 2024; Sahoo et al., 2024). For structured completion, the kernel has a direct consequence: masking removes an assignment, while replacement leaves a proposed value that may conflict with other entries. GIDD combines both forms of corruption to support revision (von Rütte et al., 2025). At inference time, confidence-based generation and remasking determine which assignments to make or revisit (Chang et al., 2022; Wang et al., 2025b), connecting the learned reconstruction problem to the evolving candidate.

Applications to reasoning include energy-based iterative refinement (Du et al., 2024) and non-autoregressive reasoning and planning with discrete diffusion (Ye et al., 2025). Corruption can also shape recurrent supervision: DIS varies the intermediate prediction targets (Asadulaev et al., 2025). Here we corrupt the candidate input while retaining the clean solution as the target, obtaining completion and correction problems from the same labeled example.

A recurrent denoiser can retain information beyond its visible token assignments. Self-conditioning feeds earlier predictions back to a diffusion model (Chen et al., 2023); Loopholing preserves deterministic latent information across discrete sampling steps (Jo et al., 2026). Looped Flows links local denoising objectives across noise levels and integrates a probability flow coupled to recurrent states at inference (Suleymanzade et al., 2026). In DRDM, several shared-core updates precede each discrete

candidate change, with latent memory retained across denoising steps. Appendix B compares the state updates.

3 Recursive Discrete Denoising

DRDM reconstructs a solution from the observed problem and an incomplete or incorrect candidate. Its recurrent predictor processes each candidate before the decoder updates its tokens (Figure 3).

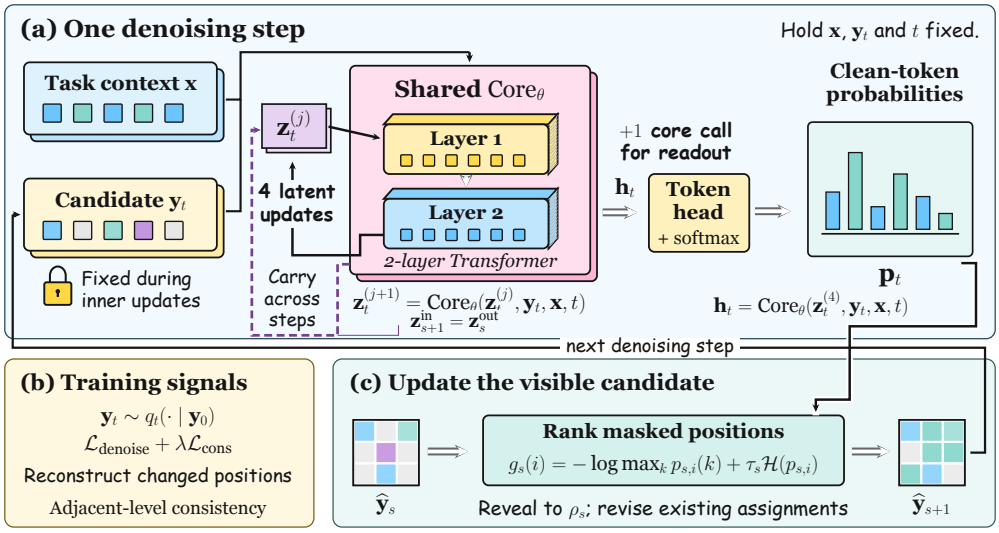


Figure 3: **Recursive discrete denoising.** (a) A shared two-layer Transformer performs four latent updates and one readout while the candidate is held fixed. Latent memory carries across denoising steps. (b) Training combines clean-token reconstruction and adjacent-level consistency. (c) Predicted probabilities guide progressive reveal and revision of editable assignments; task-fixed information is preserved. The token grids are schematic.

Problem and state. Let $\mathbf{x}$ denote the observed task context and $\mathbf{y}_0 \in \mathcal{V}^L$ the target solution, where $\mathcal{V}$ is a vocabulary of V solution tokens. The additional symbol `[MASK]` represents an unassigned coordinate. We denote the coordinates to be predicted by $\mathcal{E} \subseteq \{1, \ldots, L\}$ and any coordinates fixed by the task input by $\mathcal{I} = \{1, \ldots, L\} \setminus \mathcal{E}$. Each solver state consists of a discrete candidate $\mathbf{y}$ and real-valued latent memory $\mathbf{z}$. Corruption acts on editable solution coordinates; memory evolves through the recurrent computation. Sudoku clues, maze structure, and ARC demonstrations remain observed throughout.

3.1 Constructing Solution-Refinement Tasks

For completion and correction training, we retain some solution tokens, mask others, and replace a subset with incorrect values. We sample a corruption level t uniformly from $\{1, \ldots, T\}$ and corrupt each editable coordinate independently. Write m_t for the masking probability and u_t for the probability of replacing a token by a different token in $\mathcal{V}$. The corruption marginal for a clean token $a \in \mathcal{V}$ is

$$q_t(y_{t,i} = b \mid y_{0,i} = a) = \begin{cases} 1 - m_t - u_t, & b = a, \\ m_t, & b = \texttt{[MASK]}, \\ u_t/(V-1), & b \in \mathcal{V} \setminus \{a\}, \end{cases} \qquad i \in \mathcal{E}, \tag{1}$$

where $m_t, u_t \geq 0$ and $m_t + u_t \leq 1$. Fixed coordinates are copied unchanged. Intermediate levels mix masks and replacements; at the terminal level, $m_T = 1$ and $u_T = 0$, matching the all-mask inference initialization. Equation (1) specifies directly sampled corruption marginals.

3.2 Shared Recursive Denoiser

Following the shared-core organization of TRM (Jolicoeur-Martineau, 2025), one two-layer Transformer Core_θ serves as the denoiser at every corruption level. The level t conditions each invocation alongside the candidate $\mathbf{y}_t$ and task context $\mathbf{x}$. Starting from latent state $\mathbf{z}_t^{(0)}$, the core performs n_L updates:

$$\mathbf{z}_t^{(j+1)} = \mathrm{Core}_\theta(\mathbf{z}_t^{(j)}, \mathbf{y}_t, \mathbf{x}, t), \qquad j = 0, \ldots, n_L - 1. \tag{2}$$

The candidate is held fixed during these latent updates. The same core is then invoked for readout, followed by a lightweight token head:

$$\mathbf{h}_t = \mathrm{Core}_\theta(\mathbf{z}_t^{(n_L)}, \mathbf{y}_t, \mathbf{x}, t), \qquad \mathbf{p}_t = \mathrm{softmax}(\mathrm{Head}_\theta(\mathbf{h}_t)). \tag{3}$$

The softmax yields a clean-token distribution at each output coordinate. The base configuration uses four latent updates and one readout, totaling five invocations of the shared core per denoising step.

Latent memory is initialized from an input-dependent embedding at the first step and carried forward thereafter: $\mathbf{z}_{s+1}^{\mathrm{in}} = \mathbf{z}_s^{\mathrm{out}}$, where s indexes inference steps and in/out denote the incoming and outgoing latent states of a step. The inner recurrence therefore refines memory for a fixed candidate, while the outer recurrence updates the candidate and retains information from previous denoising steps.

3.3 Denoising and Consistency Objectives

Each corrupted candidate is paired with its clean solution, without an annotated refinement path. Let $\mathcal{M}_t = \{i \in \mathcal{E} : y_{t,i} \neq y_{0,i}\}$ be the perturbed coordinates. Using the corruption process in Section 3.1, we minimize weighted cross-entropy over these coordinates:

$$\mathcal{L}_{\mathrm{denoise}} = -\mathbb{E}_{(\mathbf{x},\mathbf{y}_0),t,\mathbf{y}_t}\left[w_t \sum_{i \in \mathcal{M}_t} \log p_{\theta,i}(y_{0,i} \mid \mathbf{x}, \mathbf{y}_t, t) \right]. \tag{4}$$

The expectation is over training examples, corruption levels, and sampled corruptions; w_t weights the loss at level t. The probabilities are produced by Equations (2)–(3), with latent-state dependence left implicit.

Standard DRDM training uses this reconstruction supervision without differentiating through the complete outer trajectory. Section 4.3 compares it with full-trajectory backpropagation while retaining the model and decoding procedure.

A consistency regularizer penalizes disagreement between clean-token distributions predicted from adjacent corruption levels:

$$\mathcal{L}_{\mathrm{total}} = \mathcal{L}_{\mathrm{denoise}} + \lambda \mathcal{L}_{\mathrm{cons}}, \qquad \mathcal{L}_{\mathrm{cons}} = \mathbb{E}[\omega(t) D_{\mathrm{KL}}(\mathbf{p}_t \,\|\, \mathbf{p}_{t-1})]. \tag{5}$$

Here λ controls the regularizer, $\omega(t)$ weights adjacent-level comparisons, and the expectation is over the sampled training pairs. Section 4.3 evaluates this regularizer.

3.4 Progressive Decoding with Token Revision

At inference, confidence determines where to add information to the candidate. Earlier editable assignments remain eligible for revision as the rest of the solution develops. Inference starts with all editable coordinates masked and all fixed information preserved. Let $\widehat{\mathbf{y}}_s$ denote the candidate before update s, so $\widehat{\mathbf{y}}_0$ is the initial candidate and $\widehat{\mathbf{y}}_R$ is the output after R updates. Each update evaluates Equations (2)–(3) on the current candidate at a descending corruption level t_s. Write $p_{s,i}$ for the resulting clean-token distribution. For each masked coordinate, let $k_i^\star = \arg\max_k p_{s,i}(k)$ and define

$$g_s(i) = -\log p_{s,i}(k_i^\star) + \tau_s \mathcal{H}(p_{s,i}), \qquad \tau_s = \tau_0 \exp(-\beta s). \tag{6}$$

Here $\mathcal{H}$ is Shannon entropy. Coordinates are ranked in ascending order of g_s: high maximum probability and low entropy yield higher priority. The coefficient τ_s anneals the entropy contribution to this deterministic ranking.

For $s = 0, \ldots, R-1$, the scheduled fraction of assigned editable coordinates is

$$\rho_s = \tfrac{1}{2}\left[1 - \cos\left(\pi \frac{s+1}{R}\right)\right]. \tag{7}$$

With $N = |\mathcal{E}|$ and $\rho_{-1} = 0$, successive targets allocate a reveal budget of $\lceil \rho_s N \rceil - \lceil \rho_{s-1} N \rceil$ positions. The decoder prioritizes remaining masks by g_s and permits revision of previously assigned editable tokens. Task-fixed coordinates are preserved. Appendix C.4 derives the scheduled counts.

The final target is $\rho_{R-1} = 1$. This deterministic policy combines a clean-token predictor with confidence-based candidate updates; Equation (1) defines the predictor's training inputs. Increasing R redistributes the reveal budget over additional recurrent updates. Time embeddings for extended horizons are obtained by linear interpolation over the training noise grid.

4 Experiments

We evaluate whether a compact recursive denoiser produces more accurate complete solutions, then examine the training choices and inference budgets behind its performance. Intermediate predictions provide a separate view of how the learned solver constructs an answer.

4.1 Experimental Setup

Tasks and evaluation. We evaluate Sudoku-Extreme and Maze-Hard from the HRM benchmark suite (Wang et al., 2025a), and ARC-AGI-1 (Chollet, 2019) and ARC-AGI-2 (Chollet et al., 2025). Sudoku and Maze require exact solutions. We report ARC task-level pass@2 on demonstration-conditioned grid transformations, using color-token grids up to 30×30. Training and evaluation are separated at the task level; model selection uses held-out validation tasks.

Comparison protocol. The compact models share task information, tokenization, and prediction targets: fixed Sudoku clues, maze structure and endpoints, and ARC demonstrations. We compare HRM, self-attention TRM, a direct Transformer, and discrete diffusion; DIS provides an intermediate-supervision comparison on ARC. A fixed-grid token mixer is evaluated separately on Sudoku. Appendix Table 14 gives complementary zero-shot results for general-purpose models.

Model and statistics. The 6.82M-parameter DRDM uses a two-layer core with four latent updates and one readout per denoising step, trained over $T = 16$ corruption levels. Unless stated otherwise, results are mean accuracy and standard deviation over three independent runs with seeds 42, 2026, and 100. Appendix A gives the representations, architecture, and optimization settings.

4.2 Accuracy on Structured Reasoning Tasks

Table 1: **Compact supervised reasoning.** Accuracy (%), mean ± SD over three seeds; ARC uses pass@2. TRM and DRDM use self-attention. HRM, TRM, and DRDM correspond to the $R = 16$ rows of Appendix Table 8. Bold denotes the highest mean, including ties.

Method	Params	Sudoku	Maze	ARC-AGI-1	ARC-AGI-2
Transformer	∼7M	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Discrete diffusion	∼7M	0.0 ± 0.0	0.7 ± 0.5	0.0 ± 0.0	0.0 ± 0.0
HRM	∼27M	56.2 ± 2.7	74.3 ± 2.1	43.7 ± 1.3	5.0 ± 0.0
TRM	∼7M	75.5 ± 1.6	84.6 ± 0.8	43.2 ± 2.1	**7.4** ± 1.6
DIS	∼7M	–	–	41.3 ± 1.7	3.4 ± 1.9
DRDM	∼7M	**85.2** ± 1.7	**87.3** ± 1.5	**50.5** ± 2.6	**7.4** ± 0.9

At approximately 7M parameters, DRDM achieves 85.2% exact accuracy on Sudoku-Extreme, 87.3% on Maze-Hard, and 50.5% pass@2 on ARC-AGI-1 (Table 1). These are improvements of 9.7, 2.7, and 7.3 percentage points over self-attention TRM. On ARC-AGI-2, both methods attain 7.4% pass@2. The gains therefore span constraint satisfaction, path construction, and demonstration-conditioned transformation, with the largest absolute improvement on Sudoku.

Using a fixed-grid token mixer. The reported TRM MLP_T token mixer is tailored to the fixed 9×9 Sudoku grid and reports 87.4% accuracy. A separate DRDM experiment with this backbone reaches 91.7% (Appendix Table 7). This comparison evaluates the same refinement formulation with a task-specific choice of core.

4.3 Training the Refinement Operator

The training study examines candidate corruption, agreement across noise levels, and gradient propagation. Tables 2 and 3 report the corresponding configurations; Appendix E specifies their objectives and execution patterns.

Table 2: **Training configurations within DRDM.** Accuracy (%), mean ± SD over three seeds. Fixed total corruption sets $\alpha_t = m_t + u_t = 1$. All rows use the summed reconstruction loss in Equation (4). Appendix Table 12 reports decoder and execution variants.

Configuration	Sudoku	Maze	ARC-AGI-1	ARC-AGI-2
Full DRDM	**85.2** ± 1.7	**87.3** ± 1.5	**50.5** ± 2.6	**7.4** ± 0.9
Fixed total corruption ($\alpha_t = 1$)	76.9 ± 1.9	81.8 ± 1.7	44.5 ± 2.5	5.0 ± 0.8
Without consistency loss	71.4 ± 2.1	78.6 ± 1.9	49.0 ± 2.4	5.7 ± 0.8

Learning from partially retained solutions. The default corruption distribution preserves some correct assignments. Setting $\alpha_t = 1$ instead masks or replaces every editable token and includes every such coordinate in the summed reconstruction loss. The two configurations thus differ in available candidate information and in the expected number $N\alpha_t$ of supervised coordinates (Appendix C.2). Training on fully corrupted candidates gives mean accuracies 8.3, 5.5, 6.0, and 2.4 percentage points below the default configuration. Reconstruction over varying degrees of completion is the more effective training formulation in this comparison.

Agreement between corruption levels. Removing the consistency objective lowers mean accuracy by 13.8 points on Sudoku and 8.7 on Maze, compared with 1.5 on ARC-AGI-1 and 1.7 on ARC-AGI-2. The largest mean drops occur on Sudoku and Maze, making adjacent-level agreement consequential on these benchmarks.

Table 3: **Training interventions in recursive models.** Accuracy (%), mean ± standard deviation over three seeds. Noise-infused TRM combines denoising with full BPTT in the TRM unroll. The final row extends gradient propagation within the complete DRDM design. Appendix Table 10 lists the configuration differences.

Configuration	Sudoku	Maze	ARC-AGI-1	ARC-AGI-2
TRM	75.5 ± 1.6	84.6 ± 0.8	43.2 ± 2.1	7.4 ± 1.6
Noise-infused TRM, full BPTT	79.8 ± 2.4	85.5 ± 1.6	46.9 ± 2.5	7.3 ± 2.2
DRDM	85.2 ± 1.7	87.3 ± 1.5	50.5 ± 2.6	7.4 ± 0.9
DRDM, full BPTT	84.0 ± 1.9	87.7 ± 1.6	49.1 ± 2.8	7.4 ± 1.6

Denoising within the TRM recurrence. Noise-infused TRM combines masked/replaced-state reconstruction with full backpropagation through time (BPTT) in the TRM backbone. This configuration improves over reference TRM by 4.3 points on Sudoku, 0.9 on Maze, and 3.7 on ARC-AGI-1 (Table 3). DRDM exceeds the combined training variant by a further 5.4, 1.8, and 3.6 points, respectively.

Gradient propagation within DRDM. Extending gradients through the full DRDM trajectory changes the four benchmark means by -1.2, $+0.4$, -1.4, and 0.0 percentage points, with the objectives and decoder unchanged. Full BPTT yields no consistent accuracy gain across these tasks. Appendix E.3 reports the additional decoder and untied-network comparisons, including their changes to parameter count and execution.

4.4 Allocating More Inference Steps

A recurrent architecture can spend additional computation on the same problem. We evaluate the same trained checkpoints at $R \in \{8, 16, 32, 64\}$ to determine where this helps. For DRDM, each budget starts a new trajectory with its reveal schedule and time conditioning rescaled to that horizon; HRM and TRM retain $H = 3, L = 6$ within their model-specific iterations.

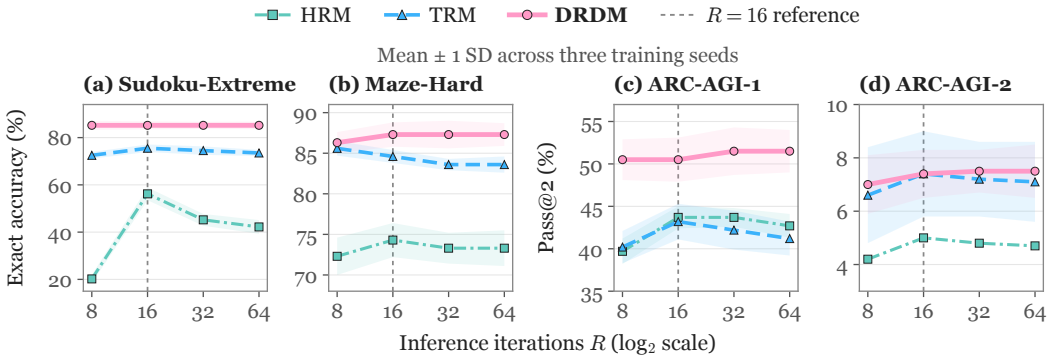


Figure 4: **Accuracy across inference horizons.** The same three checkpoints per method are evaluated at four budgets. Lines show means and bands one standard deviation; iterations retain each model's internal computation. Appendix Table 8 gives every value.

DRDM reaches 85.2% on Sudoku at eight steps and 87.3% on Maze at sixteen. Extending the horizon from sixteen to thirty-two steps adds 1.0 percentage point on ARC-AGI-1 and 0.1 on ARC-AGI-2, with the Sudoku and Maze means unchanged. All four task means then remain constant through sixty-four steps. Over sixteen to sixty-four steps, HRM's Sudoku mean changes from 56.2% to 42.2% and TRM's from 75.5% to 73.5%. Appendix D gives every evaluated mean and standard deviation.

4.5 Candidate Solutions During Refinement

The quantitative evaluation concerns completed answers. We next inspect intermediate predictions to show how candidate assignments develop within the fixed problem context. Figure 5 gives paired Sudoku and Maze examples, and Figure 6 shows a demonstration-conditioned transformation.

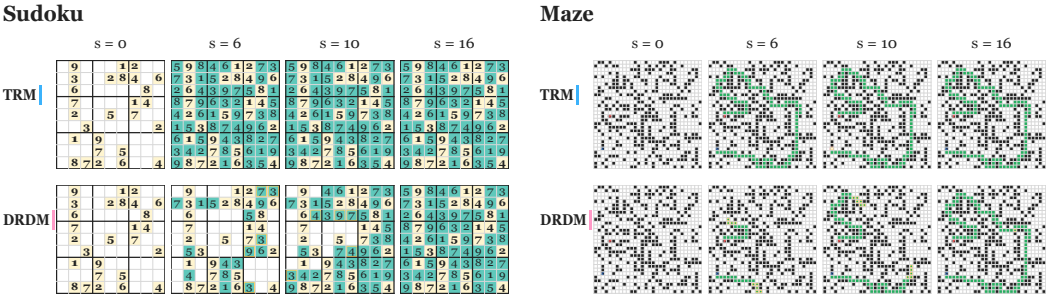


Figure 5: **Visible candidate updates in Sudoku and Maze.** Mint cells mark correct Sudoku assignments; green cells mark predicted maze paths. Columns show each model's indicated update count. The complete grids and paired sequences are enlarged in Appendix I.

The DRDM Sudoku candidates acquire assignments while retaining the observed clues, and maze candidates develop path segments within the fixed environment. In the ARC sequence, updates also reorganize the boundaries between output regions, showing how a provisional grid changes toward the demonstration-conditioned target. Appendix I provides 17 paired cases with complete grids and explanations of the visible changes.

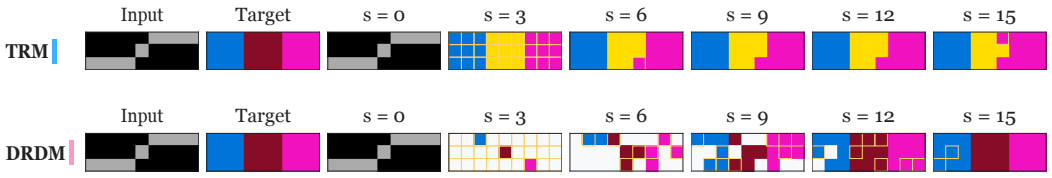


Figure 6: **Intermediate predictions for ARC task `17cae0c1`.** The $s = 0$ column repeats the input; later columns show predictions. By $s = 15$, DRDM produces the target's blue, dark-red, and magenta regions, while TRM retains a yellow center. Appendix I analyzes all 17 paired cases.

4.6 Intermediate States Along the Computation

Figure 1(a) shows DRDM approaching its terminal hidden state over successive updates, while HRM changes little until its final update. Figure 7 instead measures Sudoku candidate error against the target solution.

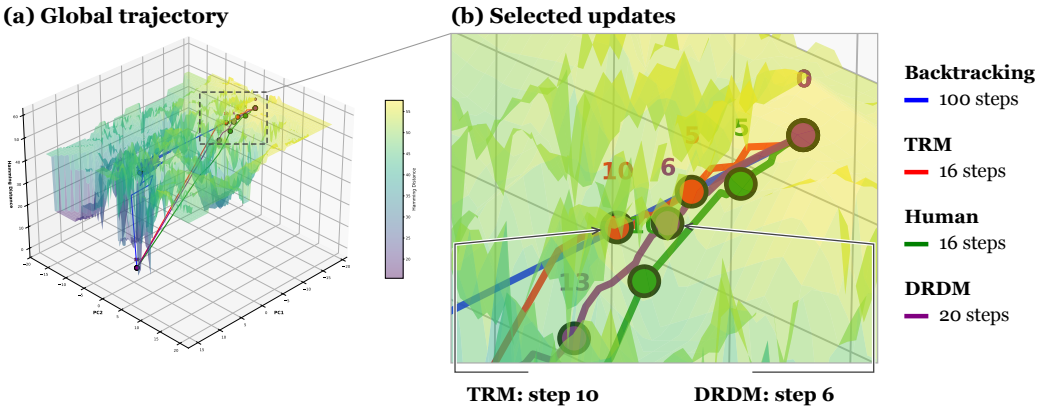


Figure 7: **Sudoku refinement in state space.** Height is target-referenced Hamming error; horizontal coordinates are PC1 and PC2. The enlarged view labels TRM step 10 and DRDM step 6 explicitly. Numbers beside the other markers are also step indices. Appendix Figure 10 provides the complete visualization.

Explicit step indices distinguish intermediate progress from eventual agreement. Candidate error counts missing and incorrect assignments; Appendix H relates this target-referenced error to distance from the final prediction.

5 Conclusion

A completed solution defines a family of refinement problems through retained, missing, and replaced assignments. DRDM uses these reconstruction problems to train a compact shared-core denoiser. This formulation improves exact solution accuracy on Sudoku-Extreme and Maze-Hard and pass@2 on ARC-AGI-1 over self-attention TRM. The formulation also supports a fixed-grid token-mixing core.

The configuration studies suggest distinct priorities for training and inference. Under the summed reconstruction objective, varying candidate completeness is more effective than corrupting every editable coordinate. Consistency across corruption levels has its largest measured effect on Sudoku and Maze, whereas propagating gradients through the entire DRDM trajectory gives no consistent accuracy gain. Inference horizons saturate at different rates: Sudoku and Maze reach their best measured means before ARC-AGI-1, and no benchmark mean improves from thirty-two to sixty-four steps. This task dependence informs how much computation to allocate to a shared-core solver.

AI USE STATEMENT

We used AI assistants for grammar checking, polishing author-written text, and routine coding assistance. The authors designed the method and experiments, made all implementation decisions affecting the reported results, verified all AI-assisted output, and take full responsibility for this paper.

REFERENCES

Arip Asadulaev, Rayan Banerjee, Fakhri Karray, and Martin Takac. Deep improvement supervision, 2025. URL https://arxiv.org/abs/2511.16886v1.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *NeurIPS*, volume 34, pp. 17981–17993, 2021.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *NeurIPS*, volume 32, 2019.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *NeurIPS*, volume 35, pp. 20232–20242, 2022.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *NeurIPS*, volume 26, 2013.

Yoshua Bengio, Éric Thibodeau-Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *ICML*, volume 32(2), pp. 226–234, 2014.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. In *NeurIPS*, volume 35, pp. 28266–28279, 2022.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. MaskGIT: Masked generative image transformer. In *CVPR*, pp. 11315–11325, 2022.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. In *ICLR*, 2023.

François Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. ARC-AGI-2: A new challenge for frontier ai reasoning systems. *arXiv preprint arXiv:2505.11831*, 2025.

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

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *ICLR*, 2019.

Yilun Du, Jiayuan Mao, and Joshua B. Tenenbaum. Learning iterative reasoning through energy diffusion. In *ICML*, volume 235, pp. 11764–11776, 2024.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. In *NeurIPS*, volume 37, pp. 133345–133385, 2024.

Khashayar Gatmiry, Nikunj Saunshi, Sashank J. Reddi, Stefanie Jegelka, and Sanjiv Kumar. Can looped transformers learn to implement multi-step gradient descent for in-context learning? In *ICML*, volume 235, pp. 15130–15152, 2024.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *NeurIPS*, volume 38, pp. 41340–41391, 2025.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-Predict: Parallel decoding of conditional masked language models. In *EMNLP-IJCNLP*, pp. 6112–6121, 2019.

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, volume 33, pp. 6840–6851, 2020.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. In *NeurIPS*, volume 34, pp. 12454–12465, 2021.

Benhao Huang, Zhengyang Geng, and Zico Kolter. Equilibrium reasoners: Learning attractors enables scalable reasoning. *arXiv preprint arXiv:2605.21488*, 2026.

Mingyu Jo, Jaesik Yoon, Justin Deschenaux, Caglar Gulcehre, and Sungjin Ahn. Loopholing discrete diffusion: Deterministic bypass of the sampling wall. In *ICLR*, 2026.

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

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *NeurIPS*, volume 35, pp. 26565–26577, 2022.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *EMNLP*, pp. 1173–1182, 2018.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *ICML*, volume 235, pp. 32819–32848, 2024.

Sajad Movahedi, Vera Milovanović, Shlomo Libo Feigin, Alexander Theus, Thomas Hofmann, Valentina Boeva, T. Konstantin Rusch, and Antonio Orvieto. Fixed-point reasoners: Stable and adaptive deep looped transformers. *arXiv preprint arXiv:2606.18206*, 2026.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *NeurIPS*, volume 38, pp. 50608–50646, 2025.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *ICLR*, 2025.

Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *NeurIPS*, volume 31, 2018.

Patrick Pynadath, Jiaxin Shi, and Ruqi Zhang. CANDI: Hybrid discrete-continuous diffusion models. *arXiv preprint arXiv:2510.22510*, 2025.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *NeurIPS*, volume 37, pp. 130136–130184, 2024.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. In *NeurIPS*, volume 34, pp. 6695–6706, 2021.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data. In *NeurIPS*, volume 37, pp. 103131–103167, 2024.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, volume 37, pp. 2256–2265, 2015.

Ayhan Suleymanzade, Chanhyuk Lee, Floor Eijkelboom, Nicholas M. Boffi, İsmail İlkan Ceylan, and Jinwoo Kim. Thinking with looped flows. *arXiv preprint arXiv:2609.11801*, 2026. URL https://arxiv.org/abs/2609.11801.

Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021. doi: 10.1016/j.patter.2021.100273.

Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The CLRS algorithmic reasoning benchmark. In *ICML*, volume 162, pp. 22084–22102, 2022.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, pp. 1096–1103, 2008.

Dimitri von Rütte, Janis Fluri, Yuhui Ding, Antonio Orvieto, Bernhard Schölkopf, and Thomas Hofmann. Generalized interpolating discrete diffusion. In *ICML*, volume 267, pp. 61810–61843, 2025.

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*, 2025a.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. In *NeurIPS*, volume 38, pp. 147282–147339, 2025b.

Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *ICLR*, 2024.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. In *ICLR*, 2025.

DRDM SUPPLEMENTARY MATERIAL

Appendix

Discrete Reasoning Diffusion Model:
Learning to Refine Partial Solutions with Recursive Discrete Denoising

Model specification, mathematical derivations, and supporting experimental results.

A Task Representation and Implementation **14**
A.1 Task-Conditioned Solution States . . . 14
A.2 Corruption and Prediction Consistency . . . 14
A.3 Base Configuration . . . 15
A.4 Inference Operations . . . 15

B State Representation and Related Formulations **16**

C Mathematical Details of Corruption and Decoding **17**
C.1 Probability Structure of the Corrupted Solution . . . 17
C.2 Which Coordinates Receive Reconstruction Supervision . . . 17
C.3 Confidence Scores and Deterministic Selection . . . 17
C.4 Scheduled Reveal Counts and Inference Budgets . . . 18

D Extended Quantitative Results **19**
D.1 Backbone-Specific Sudoku Comparison . . . 19
D.2 Accuracy at Every Evaluated Inference Horizon . . . 19
D.3 Detailed Horizon Plots . . . 20

E Supplementary Experimental Comparisons **21**
E.1 Configuration of the Training Controls . . . 21
E.2 Cross-Task Accuracy Differences . . . 21
E.3 Decoder and Execution Configurations . . . 22
E.4 Configuration Differences Across Tasks . . . 22

F General-Purpose Model Context **22**

G Sudoku Solution-Trajectory Comparison **23**
G.1 Human Trajectory Collection . . . 23
G.2 Pairwise State-Trajectory Similarity . . . 23
G.3 Geometry of Intermediate Sudoku Solutions . . . 24

H Terminal-State and Prediction Diagnostics **25**
H.1 Latent and Discrete Views of Refinement . . . 25
H.2 Decomposing Discrete Trajectory Distances . . . 26
H.3 HRM Hidden-State Diagnostics . . . 27
H.4 HRM and DRDM Sudoku Predictions . . . 28

I Qualitative Prediction Trajectories **29**
I.1 Sudoku: Filling a Coupled Constraint System . . . 30
I.2 Maze: Developing a Connected Path . . . 35
I.3 ARC: Refining Demonstration-Conditioned Transformations . . . 40

Reading routes Method A–C / Results D–F / Trajectories G–I

A Task Representation and Implementation

A.1 Task-Conditioned Solution States

Each example consists of a task context $\mathbf{x}$ and a discrete solution $\mathbf{y}_0$. Corruption is restricted to editable output coordinates. The task context remains observed throughout training and inference, and all compact supervised models preserve the task-fixed coordinates.

Sudoku-Extreme. A puzzle is a 9×9 grid of clues and empty cells. Clue cells are immutable; each remaining cell must be assigned one of nine digit values. Exact-solution accuracy requires the complete prediction to match the target grid. Structural encoding preserves the grid layout, and training, validation, and testing follow the benchmark's task-level partition.

Maze-Hard. The input specifies the maze structure and the start and goal locations. Walls and endpoints remain observed, while the model predicts the path tokens. Exact accuracy requires the full predicted path to match the target solution. All compact supervised baselines receive the same maze information through a shared input representation.

ARC-AGI. Each task provides demonstration input–output pairs and requires output grids for held-out inputs. Demonstrations form the observed context; the requested output-grid tokens are editable. Grids are serialized as discrete color arrays with maximum spatial size 30×30. Training and validation are separated at the task level, and evaluation uses the public setting. We report task-level pass@2, allowing two candidate answers per test input.

A.2 Corruption and Prediction Consistency

The total corruption probability $\alpha_t = m_t + u_t$ follows

$$\alpha_t = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min}) \sin\left(\frac{\pi t}{2T}\right). \tag{8}$$

The terminal condition $m_T = 1, u_T = 0$ gives $\alpha_T = \alpha_{\max} = 1$. The mask–replacement mixture varies with time, with both types of corruption permitted at intermediate levels. Each random replacement differs from the clean token, so $\mathcal{M}_t$ contains precisely the changed coordinates. The fixed-corruption ablation sets $\alpha_t = 1$, eliminating clean-token retention while allowing the time-dependent mixture of masks and replacements.

The consistency regularizer compares predictions of the same clean solution at adjacent corruption levels. For a sampled pair, the time-weighted penalty is

$$\ell_{\text{cons}} = \exp(-\beta_c t/T) D_{\text{KL}}(\mathbf{p}_t \,\|\, \mathbf{p}_{t-1}), \tag{9}$$

where β_c controls the decay of the consistency weight and $\mathbf{p}_t, \mathbf{p}_{t-1}$ are the clean-token predictions defined in Section 3.3. The denoising objective supplies reconstruction targets, while this regularizer encourages agreement across corruption levels. Table 2 evaluates its contribution.

A.3 Base Configuration

Table 4 summarizes the model and optimization settings. The three-seed experiments use seeds 42, 2026, and 100. The training grid contains T corruption levels, whereas R denotes the number of denoising steps allocated at inference. We evaluate $R \in \{8, 16, 32, 64\}$ in the horizon experiment.

Table 4: **Base DRDM configuration.** The core parameters are tied across all latent and readout invocations.

Setting	Value
Core	Two-layer Transformer
Parameter count	6.82M
Hidden dimension / attention heads	512 / 8
Latent updates / readout calls	4 / 1 per denoising step
Latent-memory initialization	Input-dependent, at the first step
Latent-memory propagation	Carried across denoising steps
Positional encoding	Rotary position embeddings
Forward precision	bfloat16
Training corruption levels	$T = 16$
Noise-level sampling	Uniform over $\{1, \ldots, 16\}$
Inference horizons in the sweep	8, 16, 32, 64
Time embeddings at extended horizons	Linear interpolation
Reveal fraction	Cosine schedule, Equation (7)
Entropy weight in confidence ranking	$\tau_s = 1.2 \exp(-0.15s)$
Optimizer	AdamATan2
Learning rate	10^{-4}
Warmup	2,000 steps
Learning-rate schedule	Cosine schedule with warmup
Optimizer betas	$(0.9, 0.95)$
Weight decay	1.0
Global batch size	126
Parameter EMA	Decay 0.999
Principal training seeds	42, 2026, 100

A.4 Inference Operations

Inference alternates latent computation on the current candidate with updates to that candidate (Table 5). The reveal schedule allocates new assignments while earlier predictions remain revisable. Its final target is $\rho_{R-1} = 1$. Appendix C.4 derives the schedule's integer targets and reveal budgets.

Table 5: **Operations along a DRDM inference trajectory.**

Operation	Role
Initialize	Preserve fixed context; mask editable coordinates; initialize latent memory from the input.
Refine memory	Repeatedly apply the same two-layer core with the current symbolic state held fixed.
Predict tokens	Invoke the shared core for readout and obtain clean-token distributions.
Update solution	Rank remaining masks by Equation (6); apply the scheduled reveal budget; allow provisional visible predictions to be revised.
Carry state	Retain the updated symbolic state and terminal latent memory for the next denoising step.

B State Representation and Related Formulations

The model separates observed context, candidate tokens, and recurrent memory (Table 6). During training, t indexes corruption severity and $\mathbf{y}_t$ is a corrupted solution. During decoding, s counts updates and $\widehat{\mathbf{y}}_s$ is the candidate before update s. In particular, the clean target $\mathbf{y}_0$ and initial candidate $\widehat{\mathbf{y}}_0$ are different objects.

Table 6: **Roles of the quantities in the recursive denoiser.** Corruption refers to the solution-state distribution in Equation (1).

Quantity	Meaning	Training corruption	Inference role
$\mathbf{x}$	Observed task context	Preserved	Conditions every prediction
$\mathbf{y}_t, \widehat{\mathbf{y}}_s$	Candidate solution	Masks and replacements on editable coordinates	Receives predicted solution tokens
$\mathbf{z}$	Real-valued latent memory	Only solution tokens are corrupted	Refined within a step and carried between steps

Recurrent computation. Shared transformations can learn iterative algorithms. For linear looped Transformers on Gaussian in-context regression tasks, Gatmiry et al. (2024) characterize conditions under which population-loss minimizers implement multi-step preconditioned gradient descent. This provides a concrete example of parameter sharing encoding an iterative update rule. HRM uses separately parameterized high- and low-level networks (Wang et al., 2025a); DRDM follows TRM's shared-core organization (Jolicoeur-Martineau, 2025), with four latent updates and one readout.

Intermediate supervision. Algorithmic data can provide intermediate states alongside final outputs: the CLRS benchmark exposes such states as optional supervision hints (Veličković et al., 2022). DIS constructs corrupted intermediate targets (Asadulaev et al., 2025). DRDM uses corrupted inputs with clean targets, specifying the refinement problems used for training. Appendix E describes the corresponding training controls.

Continuous state and training. CANDI combines discrete corruption with Gaussian noise in continuous token representations (Pynadath et al., 2025); DRDM corrupts solution tokens while updating memory deterministically. Loopholing trains its carry path by self-conditioning: a first pass on a noisy input supplies detached context to a second pass on that input (Jo et al., 2026). In DRDM, the shared core is also evaluated repeatedly within each inference step, before the candidate changes.

Discrete generative formulations. Absorbing diffusion connects score estimation to reconstruction: Ou et al. (2025) express its concrete score through clean-data conditional probabilities and an analytic time factor. Discrete Flow Matching constructs transition dynamics from interpolating probability paths and learned posteriors (Gat et al., 2024). These formulations distinguish the distribution of training states, the quantity predicted by the network, and the dynamics used for sampling.

Corruption and decoding. Discrete diffusion has also been applied to reasoning and planning (Ye et al., 2025). GIDD combines masking and token noise (von Rütte et al., 2025), while remasking revisits assigned tokens during generation (Wang et al., 2025b). Mixed corruption determines the reconstruction inputs used for training; revision determines which existing assignments can change during inference. Our decoder uses confidence to rank new assignments and permits revision of editable tokens.

Iterative computation and inference horizons. IRED learns annealed energy landscapes under continuous corruption (Du et al., 2024). Here, refinement is parameterized by a clean-token predictor and explicit discrete candidate updates. Equilibrium Reasoners and Fixed-Point Reasoners study recurrent stability and the allocation of inner and outer computation (Huang et al., 2026; Movahedi et al., 2026). Section 4.4 examines accuracy as the decoding horizon grows, with the reveal schedule adjusted to each horizon.

C Mathematical Details of Corruption and Decoding

This section derives probability, score, and counting identities for the formulation in Section 3.

C.1 Probability Structure of the Corrupted Solution

Fix a clean example $(\mathbf{x}, \mathbf{y}_0)$, a level t, and $N = |\mathcal{E}|$ editable coordinates. Assume $V \geq 2$, $m_t, u_t \geq 0$, and $\alpha_t = m_t + u_t \leq 1$. For any clean token a, the masses in Equation (1) sum to

$$(1 - m_t - u_t) + m_t + \sum_{b \in \mathcal{V} \setminus \{a\}} \frac{u_t}{V - 1} = 1. \tag{10}$$

Thus the distribution assigns total mass u_t to incorrect vocabulary tokens, regardless of vocabulary size. Independent sampling on editable coordinates gives the joint conditional distribution

$$q_t(\mathbf{y}_t \mid \mathbf{y}_0) = \prod_{i \in \mathcal{E}} q_t(y_{t,i} \mid y_{0,i}) \prod_{i \in \mathcal{I}} \mathbf{1}\{y_{t,i} = y_{0,i}\}. \tag{11}$$

Here $\mathbf{1}\{\cdot\}$ is the indicator function. At $t = T$, $m_T = 1, u_T = 0$ gives an all-mask editable state with fixed coordinates preserved. Corrupted states are sampled directly from the marginal at each level.

Let K_t^{mask}, K_t^{repl}, and K_t^{keep} count masked, replaced, and retained editable coordinates. Conditional on the fixed example and level,

$$\begin{aligned}(K_t^{\text{mask}}, K_t^{\text{repl}}, K_t^{\text{keep}}) &\sim \text{Multinomial}(N; m_t, u_t, 1 - \alpha_t)\,, \\ \mathbb{E}[K_t^{\text{mask}}] = N m_t, \quad \mathbb{E}[K_t^{\text{repl}}] &= N u_t, \quad \mathbb{E}[K_t^{\text{keep}}] = N(1 - \alpha_t).\end{aligned} \tag{12}$$

Because replacements exclude the clean token, $K_t = |\mathcal{M}_t| = K_t^{\text{mask}} + K_t^{\text{repl}}$ is exactly the number of changed coordinates. Consequently,

$$K_t \sim \text{Binomial}(N, \alpha_t), \qquad \mathbb{E}[K_t] = N\alpha_t, \qquad \text{Var}(K_t) = N\alpha_t(1 - \alpha_t). \tag{13}$$

Coordinate independence is conditional on t; mixing over a common random level generally couples their corruption severities.

C.2 Which Coordinates Receive Reconstruction Supervision

For the same fixed example and level, write $C_{t,i} = \mathbf{1}\{i \in \mathcal{M}_t\}$ and $\ell_{t,i} = -\log p_{\theta,i}(y_{0,i} \mid \mathbf{x}, \mathbf{y}_t, t)$. The prediction, and hence $\ell_{t,i}$, can depend on the entire corrupted state. For $\alpha_t > 0$, conditional expectation gives the exact identity

$$\mathbb{E}_{q_t}\left[w_t \sum_{i \in \mathcal{E}} C_{t,i} \ell_{t,i} \right] = w_t \alpha_t \sum_{i \in \mathcal{E}} \mathbb{E}_{q_t}[\ell_{t,i} \mid C_{t,i} = 1]. \tag{14}$$

With nonnegative weights, the summed objective weights each level by w_t and its expected number of supervised coordinates $N\alpha_t$. The conditional expectation accounts for the dependence of the prediction loss on the sampled corruption. A level with $\alpha_t = 0$ contributes zero reconstruction loss; at the all-mask terminal level, every editable coordinate receives supervision.

At levels where $\alpha_t < 1$, setting $\alpha_t = 1$ changes both factors in Equation (14): the explicit coordinate-count weight and the conditional loss under the new candidate distribution. The comparison in Table 2 therefore concerns their joint effect. This expectation identity characterizes the written objective; actual gradient magnitudes also depend on the conditional predictions and their derivatives.

C.3 Confidence Scores and Deterministic Selection

At inference step s, let $p = p_{s,i}$ be the clean-token distribution at a masked coordinate i, and let $a = \max_{k \in \mathcal{V}} p(k)$. We use natural logarithms and the convention $0 \log 0 = 0$. Then $1/V \leq a \leq 1$ and the score in Equation (6) is $g_s(i) = -\log a + \tau_s \mathcal{H}(p)$, with $\tau_s \geq 0$.

Bounds at a fixed maximum probability. Every nonzero $p(k)$ is at most a, so averaging $-\log p(k) \geq -\log a$ with weights $p(k)$ yields the lower entropy bound below. For the upper bound, choose one maximizing token and distribute the remaining mass $1-a$ over the other $V-1$ tokens. Entropy is largest when those remaining masses are equal. Therefore,

$$-\log a \;\leq\; \mathcal{H}(p) \;\leq\; h_2(a) + (1-a)\log(V-1) \;\leq\; \log V, \tag{15}$$

where $h_2(a) = -a\log a - (1-a)\log(1-a)$ is binary entropy. These bounds also cover tied maxima. Substitution gives

$$(1+\tau_s)(-\log a) \;\leq\; g_s(i) \;\leq\; -\log a + \tau_s[h_2(a) + (1-a)\log(V-1)]\,. \tag{16}$$

In particular, $0 \leq g_s(i) \leq (1+\tau_s)\log V$. A point-mass prediction has score zero, while a uniform prediction has score $(1+\tau_s)\log V$. The ranking combines preferred-token mass and vocabulary-wide uncertainty.

Comparing two coordinates. For masked coordinates i, j, write $a_i = \max_k p_{s,i}(k)$, $H_i = \mathcal{H}(p_{s,i})$, and analogously for j. Then

$$g_s(i) - g_s(j) = \log\left(\frac{a_j}{a_i}\right) + \tau_s(H_i - H_j). \tag{17}$$

If $a_i \geq a_j$ and $H_i \leq H_j$, coordinate i receives at least as high a priority as j. When these two criteria favor different coordinates, τ_s determines their relative influence. Equal maximum probabilities are ordered by entropy when $\tau_s > 0$; equal entropies are ordered by maximum probability. Given the predicted distributions and a fixed convention for resolving ties, this rule determines the selected positions without sampling from those distributions.

Effect of annealing. For a fixed prediction distribution and $\tau_s = \tau_0 \exp(-\beta s)$ with $\tau_0 \geq 0$ and $\beta > 0$,

$$0 \leq g_s(i) - (-\log a) = \tau_s \mathcal{H}(p) \leq \tau_0 \exp(-\beta s)\log V. \tag{18}$$

At fixed predictions, annealing reduces the entropy contribution according to the exponential bound in Equation (18).

C.4 Scheduled Reveal Counts and Inference Budgets

Let $R \geq 1$ be an integer inference budget and $N \geq 0$ the number of editable coordinates. For $s = 0, \ldots, R-1$, let $n_s = \lceil N\rho_s \rceil$ be the scheduled assignment target after update s, and set $n_{-1} = 0$. The incremental reveal budget is $d_s = n_s - n_{s-1}$.

Monotonic targets and the total reveal budget. The continuous schedule $f(v) = \frac{1}{2}[1 - \cos(\pi v)]$ on $0 \leq v \leq 1$ has derivative $f'(v) = \frac{\pi}{2}\sin(\pi v) \geq 0$, with $f(0) = 0$ and $f(1) = 1$. Since $\rho_s = f((s+1)/R)$, the integer targets are nondecreasing and satisfy

$$0 = n_{-1} \leq n_0 \leq \cdots \leq n_{R-1} = N, \qquad d_s \geq 0, \qquad \sum_{s=0}^{R-1} d_s = \sum_{s=0}^{R-1}(n_s - n_{s-1}) = N. \tag{19}$$

If assigned coordinates remain assigned and each update reveals d_s remaining masks, induction gives exactly n_s assigned coordinates after update s: the preceding state has n_{s-1} assignments, and $d_s \leq N - n_{s-1}$ masks are available. Under this condition, all N coordinates are assigned at the last update.

Discrete increments of the cosine schedule. Using $\rho_{-1} = 0$ and the cosine difference identity,

$$\rho_s - \rho_{s-1} = \sin\left(\frac{(2s+1)\pi}{2R}\right)\sin\left(\frac{\pi}{2R}\right). \tag{20}$$

The continuous increments are smallest near the two ends and largest near the middle of the horizon. To describe rounding, define $e_s = n_s - N\rho_s \in [0, 1)$ and $e_{-1} = 0$. Then

$$d_s = N(\rho_s - \rho_{s-1}) + e_s - e_{s-1}, \qquad \left|d_s - N(\rho_s - \rho_{s-1})\right| < 1. \tag{21}$$

Rounding can produce a step with $d_s = 0$, especially when R is large relative to N. Such a step still executes latent refinement and readout, and previously assigned values remain eligible for revision.

Changing the horizon. Writing the horizon explicitly makes its role clear:

$$n_s^{(R)} = \left\lceil \frac{N}{2} \left[1 - \cos\left(\pi \frac{s+1}{R} \right) \right] \right\rceil . \tag{22}$$

For $R' > R$ and $0 \le s < R$, $n_s^{(R')} \le n_s^{(R)}$. A larger budget therefore redistributes reveals over a longer recurrence. Each budget starts a new trajectory with its own reveal schedule and the time conditioning in Section 3.4.

With n_L latent updates and one readout per denoising step, the defined recurrence performs $R(n_L+1)$ shared-core invocations. The base setting $n_L = 4$ therefore uses $5R$ core invocations, or $10R$ Transformer-layer applications because each core contains two layers.

D Extended Quantitative Results

D.1 Backbone-Specific Sudoku Comparison

Table 7 compares self-attention and fixed-grid token mixing on Sudoku. Self-attention results are aggregated over the principal three-seed experiments. The MLP_T results come from a separate DRDM experiment and the TRM preprint, respectively, and are individual accuracy estimates.

Table 7: **Sudoku accuracy by backbone.** Self-attention results are three-seed aggregates; MLP_T results are separate accuracy estimates.

Core	Method	Accuracy (%)	Estimate / source
Self-attention	TRM	75.5 ± 1.6	Three-seed mean $\pm$ SD
Self-attention	DRDM	85.2 ± 1.7	Three-seed mean $\pm$ SD
MLP_T	TRM	87.4	Reported reference (Jolicoeur-Martineau, 2025)
MLP_T	DRDM	91.7	Separate experiment

D.2 Accuracy at Every Evaluated Inference Horizon

Table 8 gives the means and standard deviations underlying Figure 4. Each method is evaluated at all horizons using the same three independently trained checkpoints. The standard deviations quantify variation across training runs.

Table 8: **Complete horizon evaluation.** Accuracy (%), mean $\pm$ standard deviation over seeds 42, 2026, and 100. R counts model-specific inference iterations.

R	Method	Sudoku	Maze	ARC-AGI-1	ARC-AGI-2
8	HRM	20.2 ± 1.7	72.3 ± 2.3	39.7 ± 1.5	4.2 ± 0.0
8	TRM	72.5 ± 1.4	85.6 ± 0.9	40.2 ± 1.9	6.6 ± 1.8
8	DRDM	85.2 ± 1.9	86.3 ± 1.3	50.5 ± 2.4	7.0 ± 1.1
16	HRM	56.2 ± 2.7	74.3 ± 2.1	43.7 ± 1.3	5.0 ± 0.0
16	TRM	75.5 ± 1.6	84.6 ± 0.8	43.2 ± 2.1	7.4 ± 1.6
16	DRDM	85.2 ± 1.7	87.3 ± 1.5	50.5 ± 2.6	7.4 ± 0.9
32	HRM	45.2 ± 2.5	73.3 ± 1.9	43.7 ± 1.1	4.8 ± 0.0
32	TRM	74.5 ± 1.8	83.6 ± 0.7	42.2 ± 2.3	7.2 ± 1.4
32	DRDM	85.2 ± 1.5	87.3 ± 1.7	51.5 ± 2.8	7.5 ± 0.8
64	HRM	42.2 ± 2.9	73.3 ± 2.2	42.7 ± 1.4	4.7 ± 0.0
64	TRM	73.5 ± 1.3	83.6 ± 1.0	41.2 ± 2.0	7.1 ± 1.5
64	DRDM	85.2 ± 1.6	87.3 ± 1.4	51.5 ± 2.5	7.5 ± 1.0

Table 9 reports the change in mean accuracy from $R = 16$ to $R = 64$, in percentage points. HRM and TRM use $H = 3, L = 6$ in this sweep; Movahedi et al. (2026) report increasing TRM accuracy with additional outer refinement segments under a different inner recurrence.

Table 9: **Change in mean accuracy from 16 to 64 iterations**, in percentage points.

Method	Sudoku	Maze	ARC-AGI-1	ARC-AGI-2
HRM	−14.0	−1.0	−1.0	−0.3
TRM	−2.0	−1.0	−2.0	−0.3
DRDM	0.0	0.0	+1.0	+0.1

D.3 Detailed Horizon Plots

Figure 8 pairs the horizon sweeps with 16-versus-64 comparisons from Tables 8–9. Shading and error bars show one standard deviation across the same three training seeds.

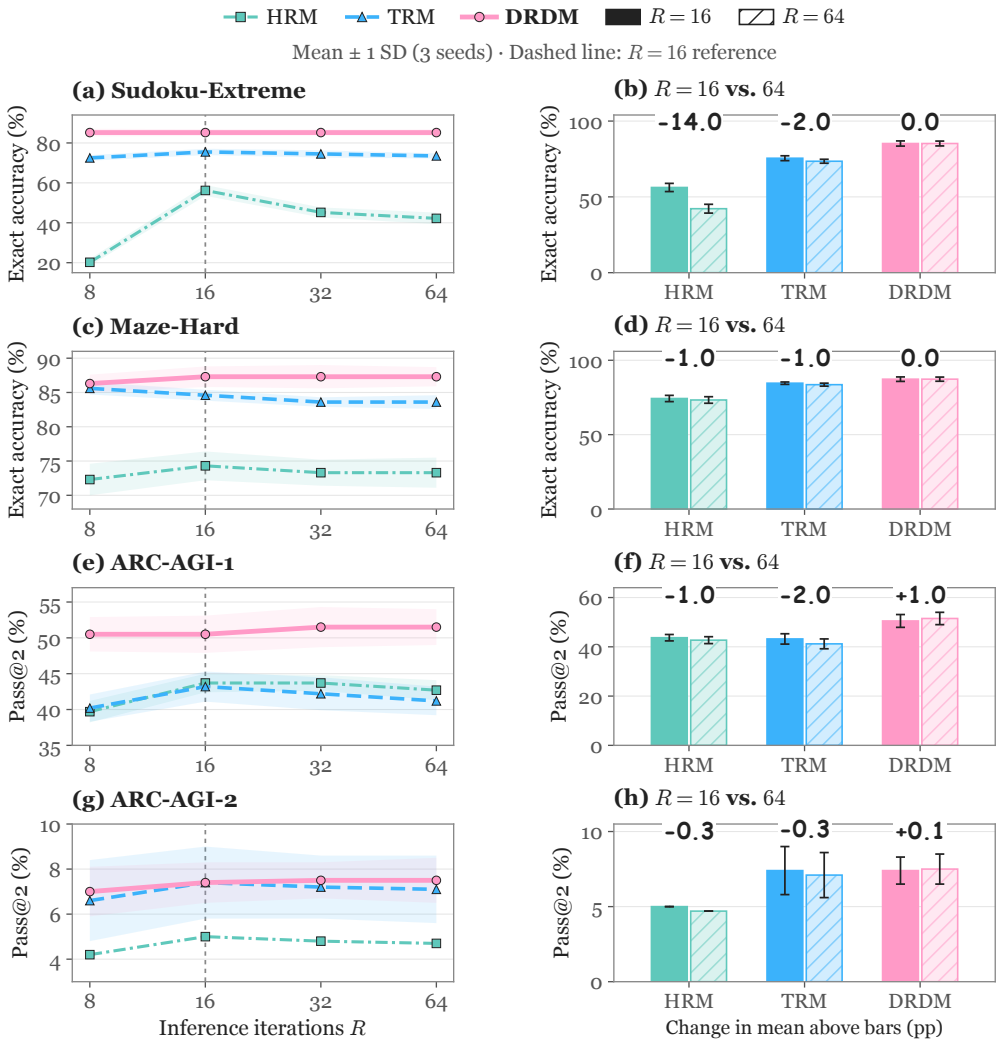


Figure 8: **Horizon sweeps and extended-inference comparisons.** Left: the complete $R \in \{8, 16, 32, 64\}$ sweep. Right: mean accuracy at 16 (solid) and 64 (hatched) iterations; labels above bars give the change in percentage points. Both views use the same three-seed results as Figure 4. Iterations retain each model's internal execution schedule.

E SUPPLEMENTARY EXPERIMENTAL COMPARISONS

This section gives the training-control configurations, additional decoder and execution comparisons, and all four-task mean differences underlying Section 4.3. Absolute accuracies and across-seed standard deviations appear in Tables 1–3 and 12.

E.1 CONFIGURATION OF THE TRAINING CONTROLS

Noise-infused TRM introduces corrupted-state reconstruction and full BPTT into the TRM unroll (Table 10). Its solution-state objective is reconstruction; recurrent execution and decoding follow TRM. The full-BPTT DRDM variant retains the DRDM objectives and decoder while extending gradient propagation through the complete latent-refinement and readout trajectory.

Table 10: **Configuration distinctions between the training controls.** All four configurations use self-attention. Standard training denotes the corresponding reference configuration; full BPTT differentiates through the complete recursive trajectory. Each control uses its corresponding TRM or DRDM execution schedule.

Configuration	Solution-state supervision	Recurrent execution	Gradient propagation
TRM	Reference TRM training	TRM recursive unroll	Standard TRM training
Noise-infused TRM	Corrupted-state input; clean targets on perturbed coordinates	TRM recursive unroll	Full BPTT
DRDM	Denoising and consistency objectives	Shared latent/readout schedule	Standard DRDM training
DRDM, full BPTT	Same objectives as DRDM	Same schedule as DRDM	Full BPTT through latent refinement and readout

E.2 CROSS-TASK ACCURACY DIFFERENCES

Table 11 compares mean accuracy across the four tasks. For configurations A and B, an entry is $\Delta(A, B) = \bar{a}_A - \bar{a}_B$, where $\bar{a}$ is mean accuracy in percent. Positive entries favor the first configuration. All differences are expressed in percentage points, including ARC task-level pass@2.

Table 11: **Differences in mean accuracy across all four tasks.** Entries are arithmetic differences of the means in Tables 1 and 3, in percentage points. The DRDM–DIS comparison covers the ARC tasks.

Comparison ($A - B$)	Sudoku	Maze	ARC-AGI-1	ARC-AGI-2
DRDM − TRM	+9.7	+2.7	+7.3	0.0
DRDM − HRM	+29.0	+13.0	+6.8	+2.4
DRDM − DIS	–	–	+9.2	+4.0
Noise-infused TRM − TRM	+4.3	+0.9	+3.7	−0.1
DRDM − noise-infused TRM	+5.4	+1.8	+3.6	+0.1
DRDM, full BPTT − DRDM	−1.2	+0.4	−1.4	0.0

Statistical convention. The principal experiments summarize seeds 42, 2026, and 100; full-BPTT DRDM follows the same protocol. Standard deviations measure variation across training runs. The horizon sweep evaluates the same checkpoints at each budget. Differences in mean accuracy are computed from the reported rounded values.

E.3 Decoder and Execution Configurations

Table 12 reports two alternative decoder and execution configurations.

Table 12: **Decoder and execution configurations.** Accuracy (%), mean ± SD over three seeds. The first row repeats the reference DRDM results.

Configuration	Sudoku	Maze	ARC-AGI-1	ARC-AGI-2
Full DRDM	85.2 ± 1.7	87.3 ± 1.5	50.5 ± 2.6	7.4 ± 0.9
Non-progressive decoding	66.9 ± 2.3	80.0 ± 1.8	39.3 ± 2.4	2.8 ± 0.5
Untied feed-forward stages	4.8 ± 0.7	20.0 ± 1.8	28.8 ± 2.2	0.2 ± 0.2

Non-progressive decoding. The decoding comparison replaces the progressive scheduler with a non-progressive variant. Its reported configuration does not specify the alternative reveal, ranking, and revision operations separately, so the contrast concerns the decoder as a whole.

Untied execution. The feed-forward variant uses distinct parameters for successive stages, each executed once, in place of the repeatedly invoked shared core. Depth and FLOPs are matched in the reported comparison. Parameter count and the periodic latent/readout execution pattern change together.

E.4 Configuration Differences Across Tasks

Table 13: **Configuration changes in mean accuracy.** Each entry subtracts the full-DRDM mean from the variant mean in Tables 2 and 12, in percentage points.

Variant − full DRDM	Sudoku	Maze	ARC-AGI-1	ARC-AGI-2
Fixed corruption ($\alpha_t = 1$)	−8.3	−5.5	−6.0	−2.4
Without consistency loss	−13.8	−8.7	−1.5	−1.7
Non-progressive decoding	−18.3	−7.3	−11.2	−4.6
Untied feed-forward stages	−80.4	−67.3	−21.7	−7.2

F General-Purpose Model Context

Table 14 reports zero-shot results for pretrained general-purpose models. Table 1 evaluates compact models trained on task-specific data under a shared representation and evaluation protocol.

The table includes LLaDA-8B, a pretrained masked-diffusion language model (Nie et al., 2025). Sudoku and Maze are evaluated by exact-solution accuracy, and the two ARC columns use task-level pass@2.

Table 14: **Zero-shot results for general-purpose models.** Accuracy (%) on Sudoku and Maze; task-level pass@2 on ARC.

Method	Sudoku	Maze	ARC-AGI-1	ARC-AGI-2
DeepSeek R1	0.0	0.0	15.8	1.3
Claude 4.5	12.3	2.4	47.7	7.8
GPT-5.2	24.0	20.7	55.7	9.7
Gemini 2.5 Pro	6.3	5.1	37.0	4.9
Grok-4-thinking	3.8	2.1	66.7	16.0
Bespoke (Grok-4)	7.4	8.2	79.6	29.4
LLaDA-8B	14.5	0.0	14.7	0.0

G SUDOKU SOLUTION-TRAJECTORY COMPARISON

We compare Sudoku trajectories through state-representation similarity and Hamming distance to the target solution. Appendix I shows the grid predictions.

G.1 HUMAN TRAJECTORY COLLECTION

Five middle-school students solved a collection of 50 Sudoku puzzles through a web interface that logged each decision and its timestamp. Participants could solve overlapping subsets of puzzles, yielding 100 trajectories. The protocol received a no-risk determination, and the interaction logs were anonymized. These traces provide the human reference for the trajectory comparisons.

G.2 PAIRWISE STATE-TRAJECTORY SIMILARITY

For trajectories $\mathcal{T}_a = (\mathbf{y}_1^a, \ldots, \mathbf{y}_{T_a}^a)$ and $\mathcal{T}_b = (\mathbf{y}_1^b, \ldots, \mathbf{y}_{T_b}^b)$, we average cosine similarities between state representations at corresponding indices:

$$S(\mathcal{T}_a, \mathcal{T}_b) = \frac{1}{K} \sum_{t=1}^{K} \frac{\phi(\mathbf{y}_t^a)^\top \phi(\mathbf{y}_t^b)}{\|\phi(\mathbf{y}_t^a)\|_2 \, \|\phi(\mathbf{y}_t^b)\|_2}, \qquad K = \min(T_a, T_b), \tag{23}$$

where ϕ denotes the state representation. Each pair is aligned by its first K record indices, which count human actions or model updates. In the pairwise matrix in Figure 9, DRDM–human similarity is 0.700, compared with 0.380 for TRM–human and 0.300 for backtracking–human; the TRM–DRDM similarity is 0.460.

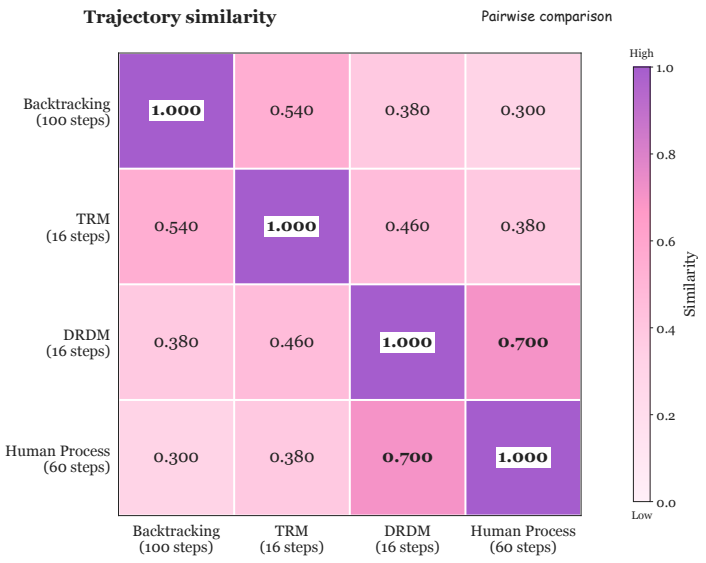


Figure 9: **Pairwise Sudoku trajectory similarity**, computed using Equation (23). Trajectory lengths are 100 steps for backtracking, 16 for TRM and DRDM, and 60 for the human process. Each off-diagonal entry compares the intermediate states of the indicated pair of trajectories.

G.3 Geometry of Intermediate Sudoku Solutions

Figure 10 compares representative backtracking, TRM, human, and DRDM trajectories. The height measures Hamming error against the target solution: each unassigned or incorrectly assigned coordinate contributes one unit. In this example, the error decreases more gradually along the DRDM trajectory than along the backtracking trajectory. The lower panels identify the methods and magnify intermediate states.

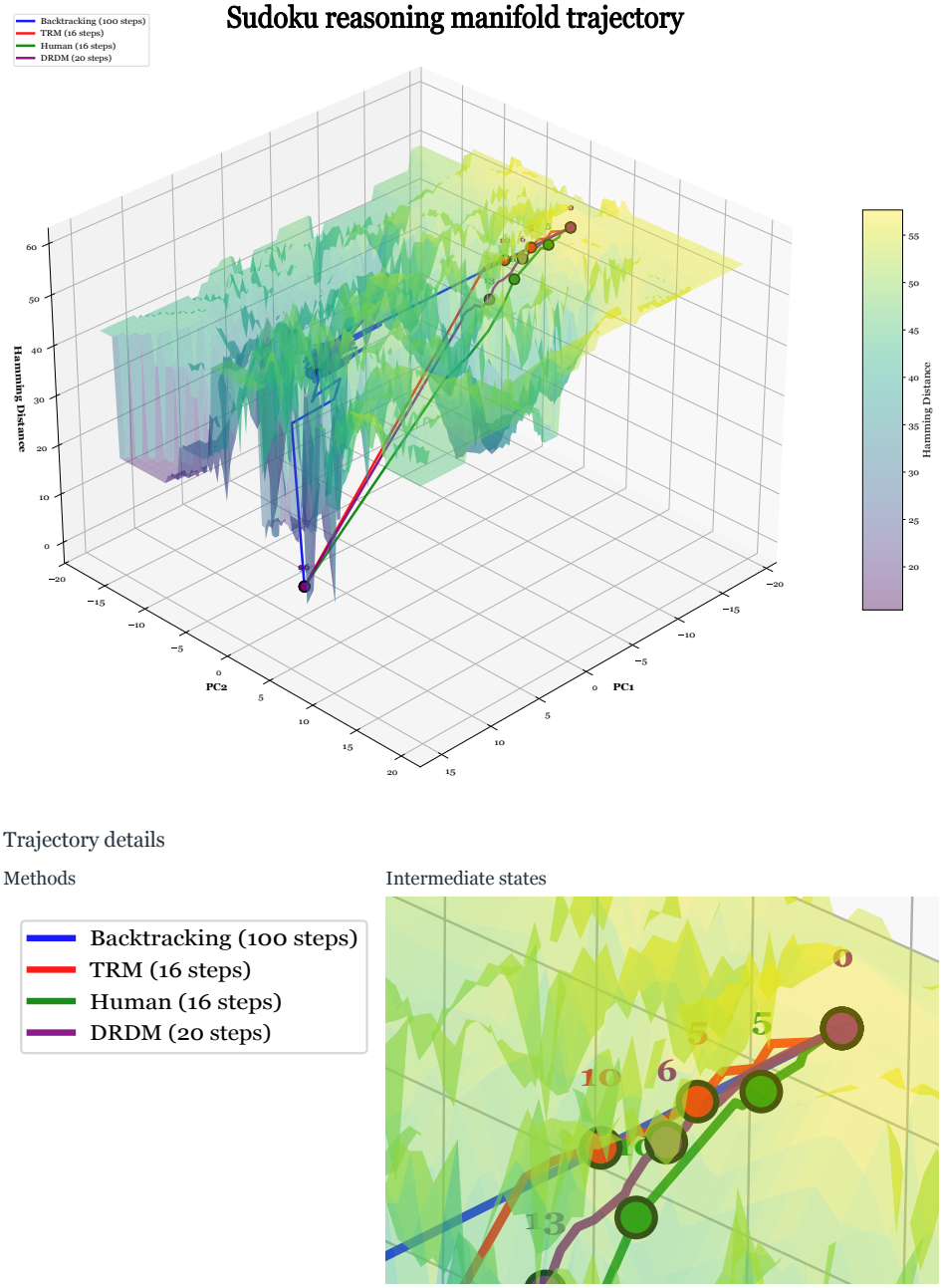


Figure 10: **Sudoku trajectories in state space.** The plotted trajectory states have horizontal coordinates PC1 and PC2 and height equal to Hamming distance to the target solution. Colored paths and numbered checkpoints indicate successive states, with lengths given in the legend. Lower panels enlarge details of the same visualization.

H TERMINAL-STATE AND PREDICTION DIAGNOSTICS

H.1 LATENT AND DISCRETE VIEWS OF REFINEMENT

We examine latent residual distance to the terminal hidden state and Hamming distance to the terminal prediction. Write $\widehat{\mathbf{y}}_s$ for a decoded prediction and $\mathbf{y}_0$ for the clean target. Let $\mathbf{z}_R$ and $\widehat{\mathbf{y}}_R$ denote the terminal states. For a trajectory with a fully assigned terminal prediction, define

$$r_s = d_z(\mathbf{z}_s, \mathbf{z}_R), \qquad h_s = d_H(\widehat{\mathbf{y}}_s, \widehat{\mathbf{y}}_R), \tag{24}$$

where d_z is the hidden-state distance used in the residual diagnostic and d_H counts disagreeing output coordinates. Thus $r_R = h_R = 0$. Every remaining mask contributes one disagreement to h_s; the distance also includes disagreements among assigned tokens.

Figure 11 compares TRM and DRDM in both spaces. In the plotted sequence, the DRDM latent residual follows an overall downward trend, whereas the TRM residual changes little until the final update. The right panel shows a steady reduction in DRDM Hamming distance to its final decoded state, while TRM remains at a similar distance until late in the sequence.

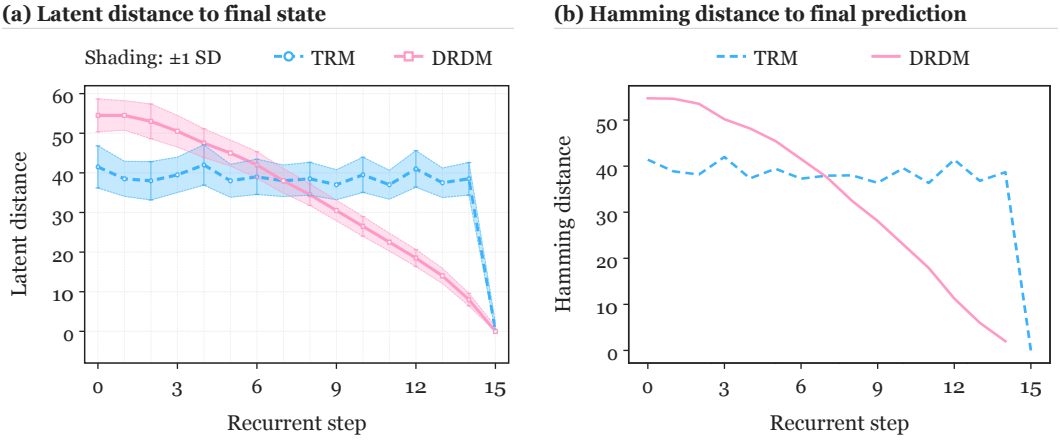


Figure 11: **Refinement relative to the terminal state.** Left: latent residual distance for TRM and DRDM, with one-standard-deviation shading. Right: Hamming distance from each decoded state to the final prediction. Each panel shows its indicated step range. Both distances reference each process's own endpoint, as defined in Equation (24).

H.2 Decomposing Discrete Trajectory Distances

The discrete diagnostic admits a decomposition that separates the number of unassigned coordinates from disagreement among assigned tokens. Let

$$\mathcal{U}_s = \{i \in \mathcal{E} : \widehat{y}_{s,i} = \texttt{[MASK]}\}, \qquad \mathcal{A}_s = \mathcal{E} \setminus \mathcal{U}_s, \qquad c_s = \sum_{i \in \mathcal{A}_s} \mathbf{1}\{\widehat{y}_{s,i} \neq \widehat{y}_{R,i}\}. \tag{25}$$

Here $\mathcal{U}_s$ contains the remaining masks, $\mathcal{A}_s$ contains assigned editable coordinates, and c_s counts their disagreement with the terminal prediction. Task-fixed coordinates contribute zero because their values are preserved. Since $\widehat{\mathbf{y}}_R$ is fully assigned and $\texttt{[MASK]} \notin \mathcal{V}$, every coordinate in $\mathcal{U}_s$ contributes exactly one disagreement. Splitting the Hamming sum over the two sets gives

$$h_s = |\mathcal{U}_s| + c_s. \tag{26}$$

Between successive predictions, the same identity gives

$$h_{s+1} - h_s = (|\mathcal{U}_{s+1}| - |\mathcal{U}_s|) + (c_{s+1} - c_s). \tag{27}$$

The first term is the net change in the number of masks. If an update reveals Δ_s coordinates and introduces no new masks, this term equals $-\Delta_s$. The second term records changes in disagreement among assigned tokens, including newly assigned coordinates and revisions to earlier predictions. Figure 11 plots the terminal-state distance $h_s = |\mathcal{U}_s| + c_s$.

Relation to target-solution error. Let $e_s = d_H(\widehat{\mathbf{y}}_s, \mathbf{y}_0)$ and let b_s count incorrectly assigned coordinates. The same partition gives

$$e_s = |\mathcal{U}_s| + b_s, \qquad b_s = \sum_{i \in \mathcal{A}_s} \mathbf{1}\{\widehat{y}_{s,i} \neq y_{0,i}\}. \tag{28}$$

Thus target Hamming error combines incomplete coverage with incorrect assignments, whereas h_s combines incomplete coverage with disagreement from the terminal prediction. Writing $e_R = d_H(\widehat{\mathbf{y}}_R, \mathbf{y}_0)$, the triangle inequality gives

$$|e_s - e_R| \leq h_s \leq e_s + e_R, \qquad \max\{0, h_s - e_R\} \leq e_s \leq h_s + e_R. \tag{29}$$

For example, the upper bound on e_s follows by inserting $\widehat{\mathbf{y}}_R$ between $\widehat{\mathbf{y}}_s$ and $\mathbf{y}_0$; the reverse bound follows by exchanging the two reference states. If the terminal prediction is correct, $e_R = 0$ and $h_s = e_s$. For an incorrect terminal prediction, $h_R = 0$ while e_R remains positive. This distinction explains why target-referenced and terminal-referenced plots measure different properties of a trajectory.

H.3 HRM Hidden-State Diagnostics

Figure 12 presents residual-distance curves for HRM and DRDM alongside an illustrative view of their trajectory geometry. Panel (a) references each trajectory's terminal hidden state. Panel (b) schematically depicts the paths and the extended-horizon segment on axes labeled PC1 and PC2. Figure 13 separately shows the corresponding visible predictions.

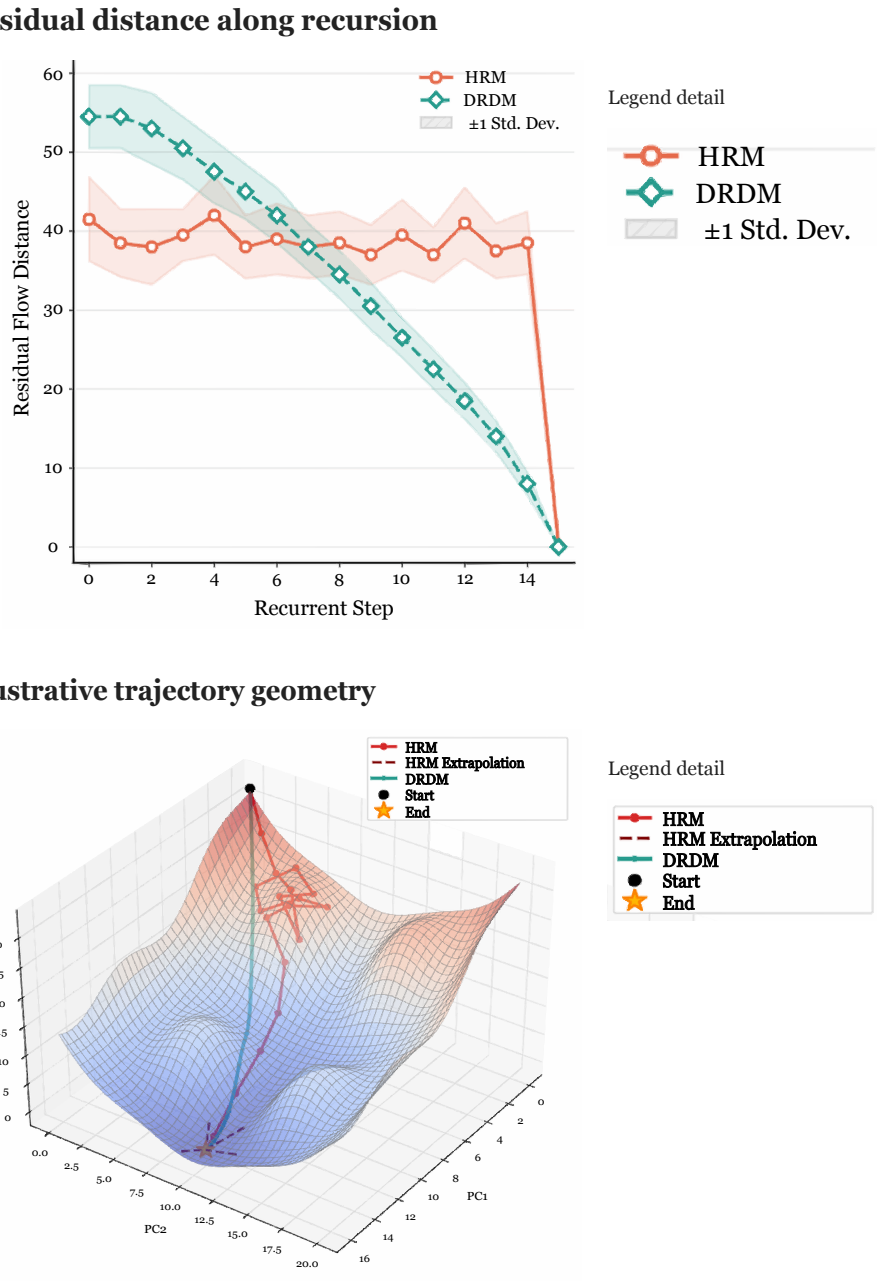


Figure 12: **Latent dynamics for HRM and DRDM.** (a) Residual distance to each trajectory's terminal hidden state across recursive updates. (b) Illustrative trajectory geometry, including the extended-horizon segment. Corresponding Sudoku predictions appear in Figure 13.

H.4 HRM AND DRDM SUDOKU PREDICTIONS

Figure 13 compares predicted Sudoku grids at seven stages. The final column shows the extended-horizon comparison at step 18. Cell colors indicate agreement with the target. Aggregate accuracy across inference budgets appears in Figure 4.

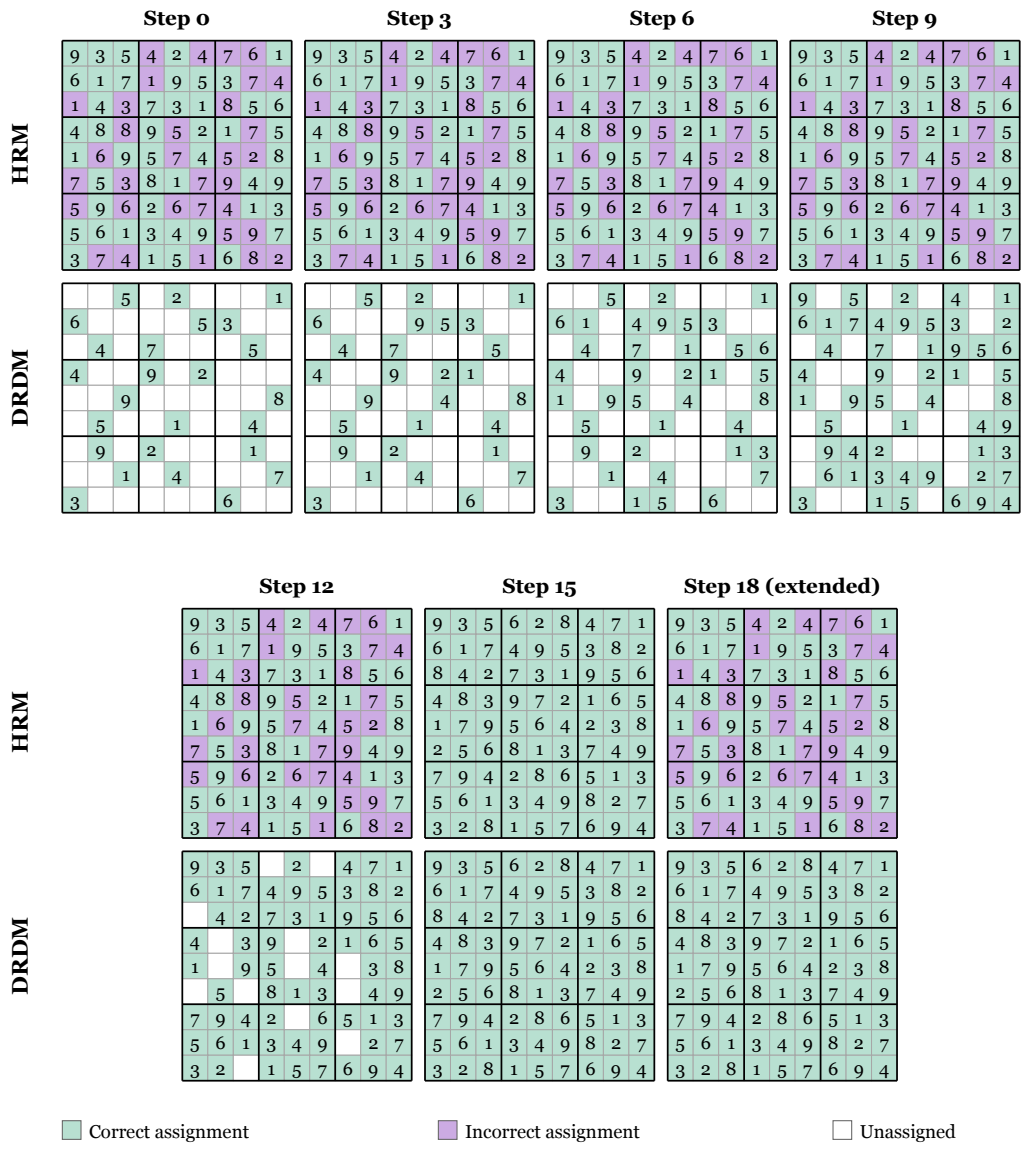
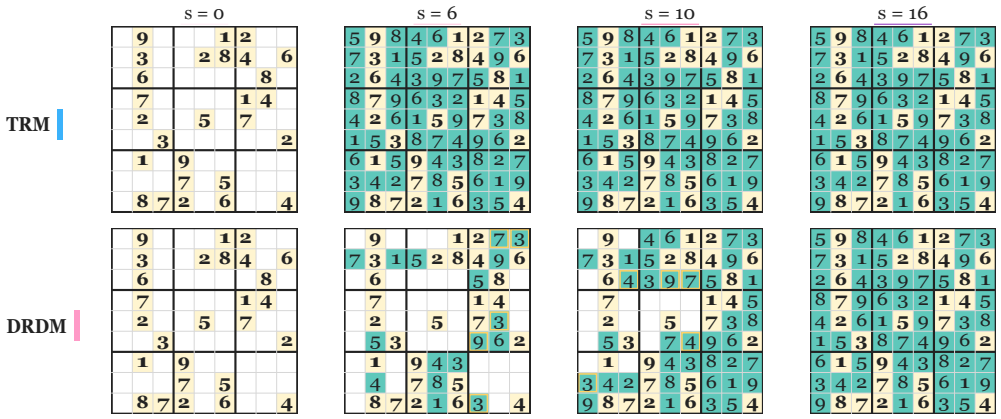


Figure 13: **Sudoku predictions for HRM and DRDM.** The upper block shows steps 0, 3, 6, and 9; the lower block shows steps 12, 15, and 18 (extrapolation). Mint and purple cells respectively agree and disagree with the target solution; blank cells are unassigned.

I Qualitative Prediction Trajectories

The following 17 cases comprise five Sudoku puzzles, five mazes, and seven ARC tasks. Adjacent rows compare models at the indicated update counts; each step is a model-specific update. Each case is followed by an analysis of the observed spatial and temporal changes. Sudoku and Maze display steps 0, 3, and 6 in the upper block, followed by 10, 13, and 16 in the lower block.

Sudoku



Maze

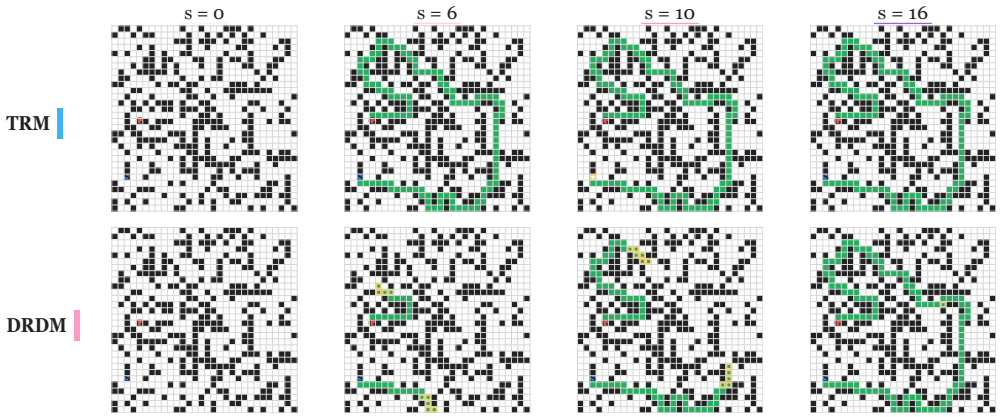


Figure 14: **Intermediate solution states in Sudoku and Maze.** Paired examples show predictions at four inference stages. In Sudoku, bold digits are clues, mint cells are correct predictions, and blank cells are unassigned; green marks predicted paths in Maze. The complete sequences appear in the case studies that follow.

The two examples highlight different spatial units of refinement. In Sudoku, DRDM expands a set of assigned cells within a fixed clue grid; the TRM row already displays a complete correct prediction at $s = 6$. In Maze, early DRDM predictions form local path segments that extend through the maze at later stages. The following case studies trace these changes through six displayed stages and then examine object and region transformations in ARC.

I.1 SUDOKU: FILLING A COUPLED CONSTRAINT SYSTEM

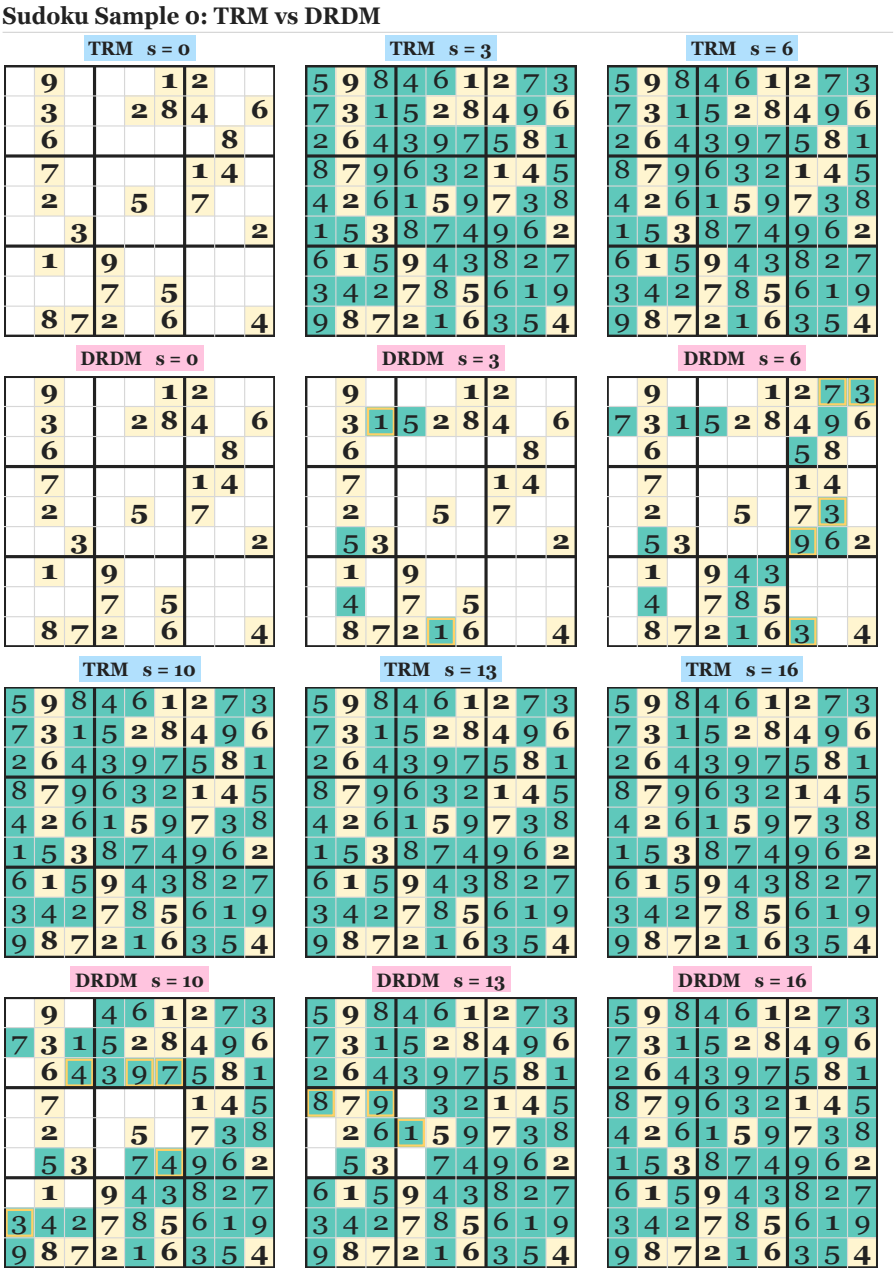


Figure 15: **Sudoku sample 0: completing the central band.**

DRDM's early assignments are scattered across the second row and lower rows. By $s = 10$, the top and bottom bands are densely populated, while the middle band retains several gaps. Those central gaps shrink at $s = 13$ and disappear at $s = 16$, with the displayed assignments remaining mint throughout. TRM already presents a fully mint grid at $s = 3$ and preserves it thereafter. The paired grids contrast early completion with a spatially ordered sequence of partial assignments.

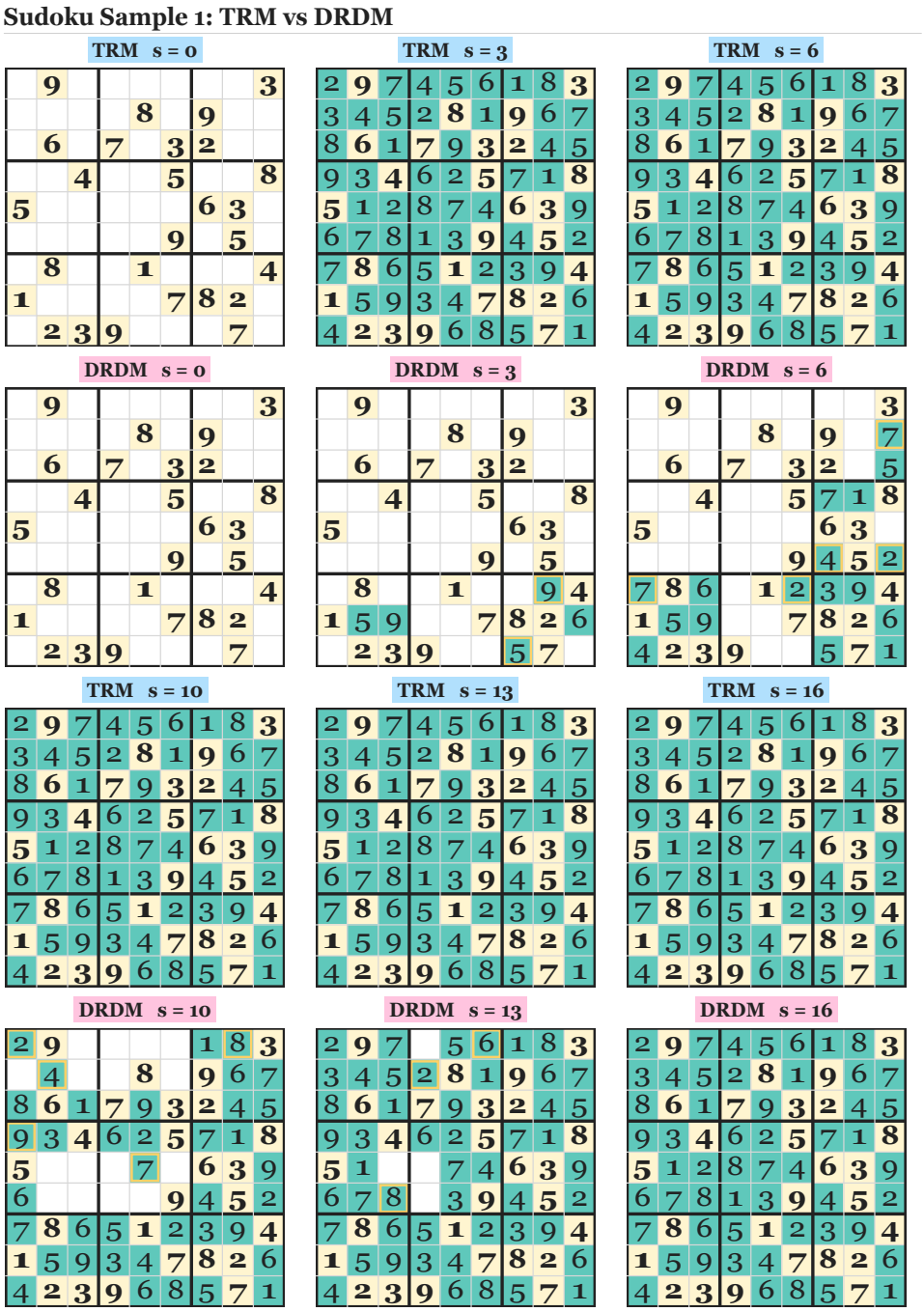


Figure 16: **Sudoku sample 1: expanding from the lower rows.**

DRDM first fills entries in the bottom three rows at $s = 3$, then extends through the right side and lower band at $s = 6$. The upper-left region and central cells remain less populated at $s = 10$; a few gaps survive at $s = 13$ before the grid becomes fully mint at $s = 16$. TRM displays the same correct completion from $s = 3$ onward. The sequence highlights a strongly uneven spatial order of assignment within a globally coupled constraint system.

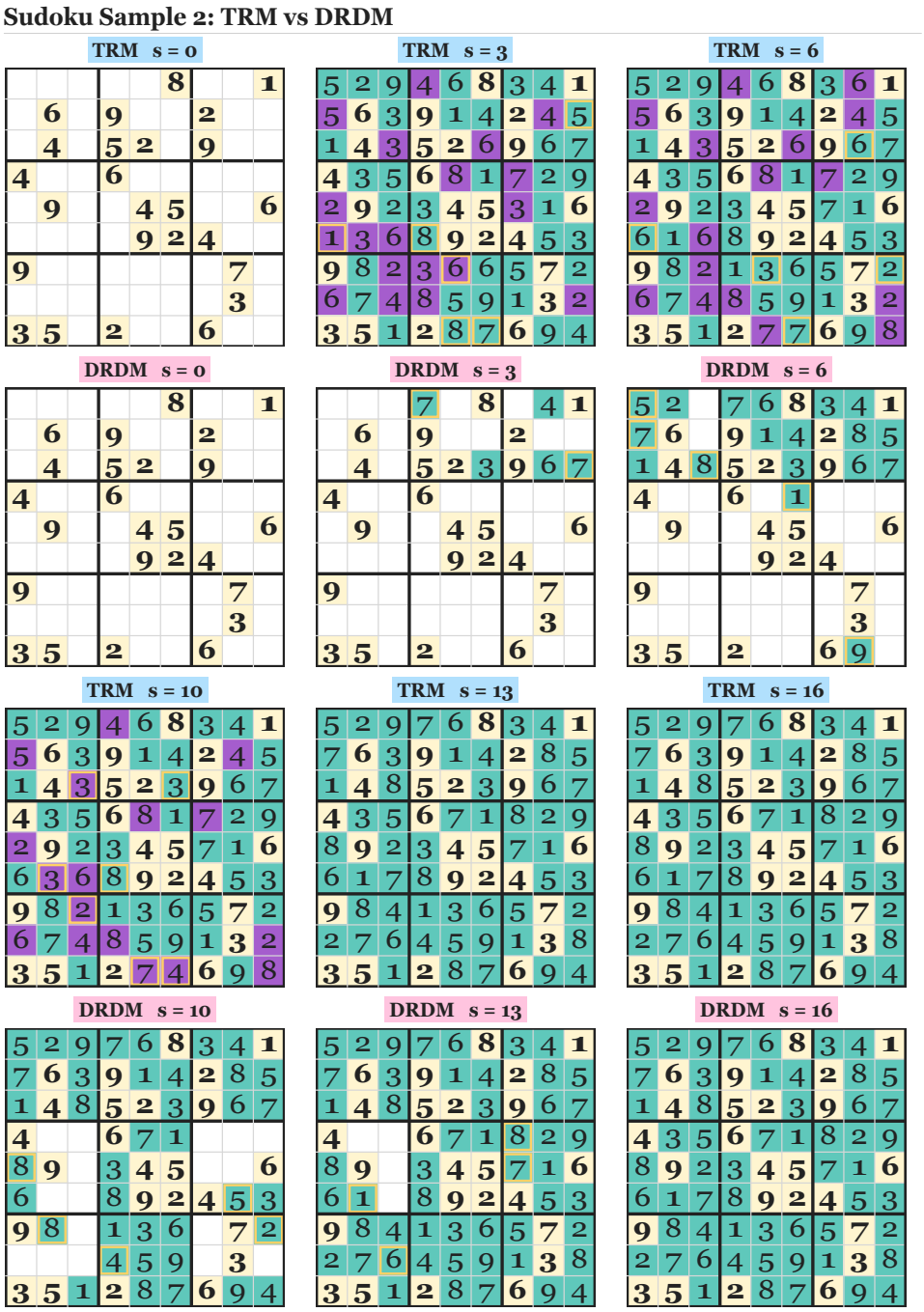


Figure 17: **Sudoku sample 2: upper-band completion and TRM error correction.**

DRDM concentrates its initial assignments in the upper band, which is largely filled by $s = 6$ while most lower cells remain blank. At $s = 10$, coverage extends into the bottom band, leaving gaps around the center. TRM's early dense grids contain purple errors in both upper and lower regions, including several bottom-row entries at $s = 10$; these disappear by $s = 13$. DRDM completes its remaining gaps at $s = 16$, providing a contrasting progression through consistently mint partial grids.

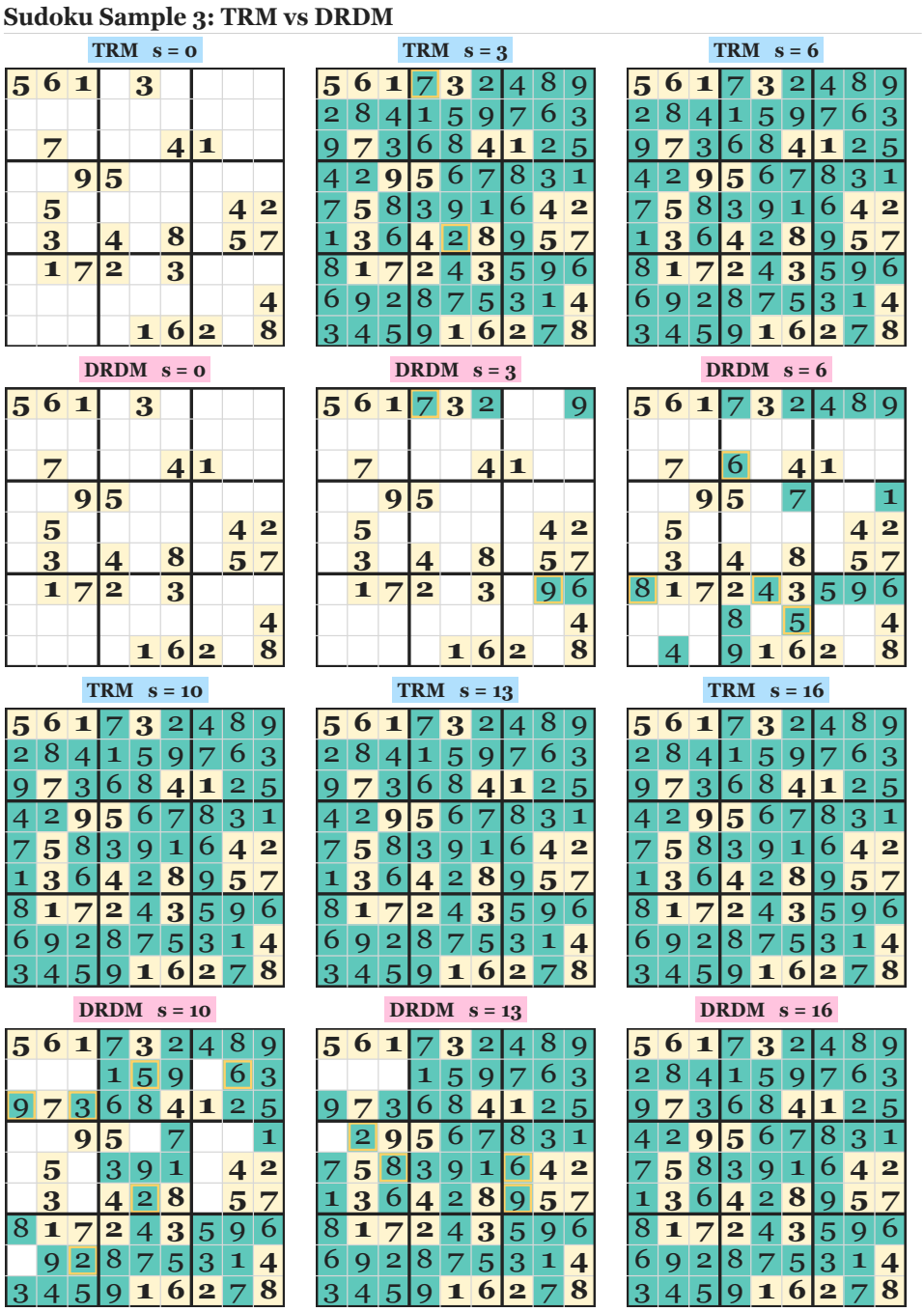


Figure 18: **Sudoku sample 3: filling separated rows before the interior.**

The first and seventh rows become fully populated in DRDM's $s = 6$ panel, although much of the intervening grid remains blank. By $s = 10$, the lower band is nearly complete and the middle band still contains several gaps. At $s = 13$, unresolved entries are concentrated toward the left side, including the beginning of the second row; $s = 16$ is fully mint. TRM's stable mint completion from $s = 3$ provides a reference for this nonuniform assignment pattern.

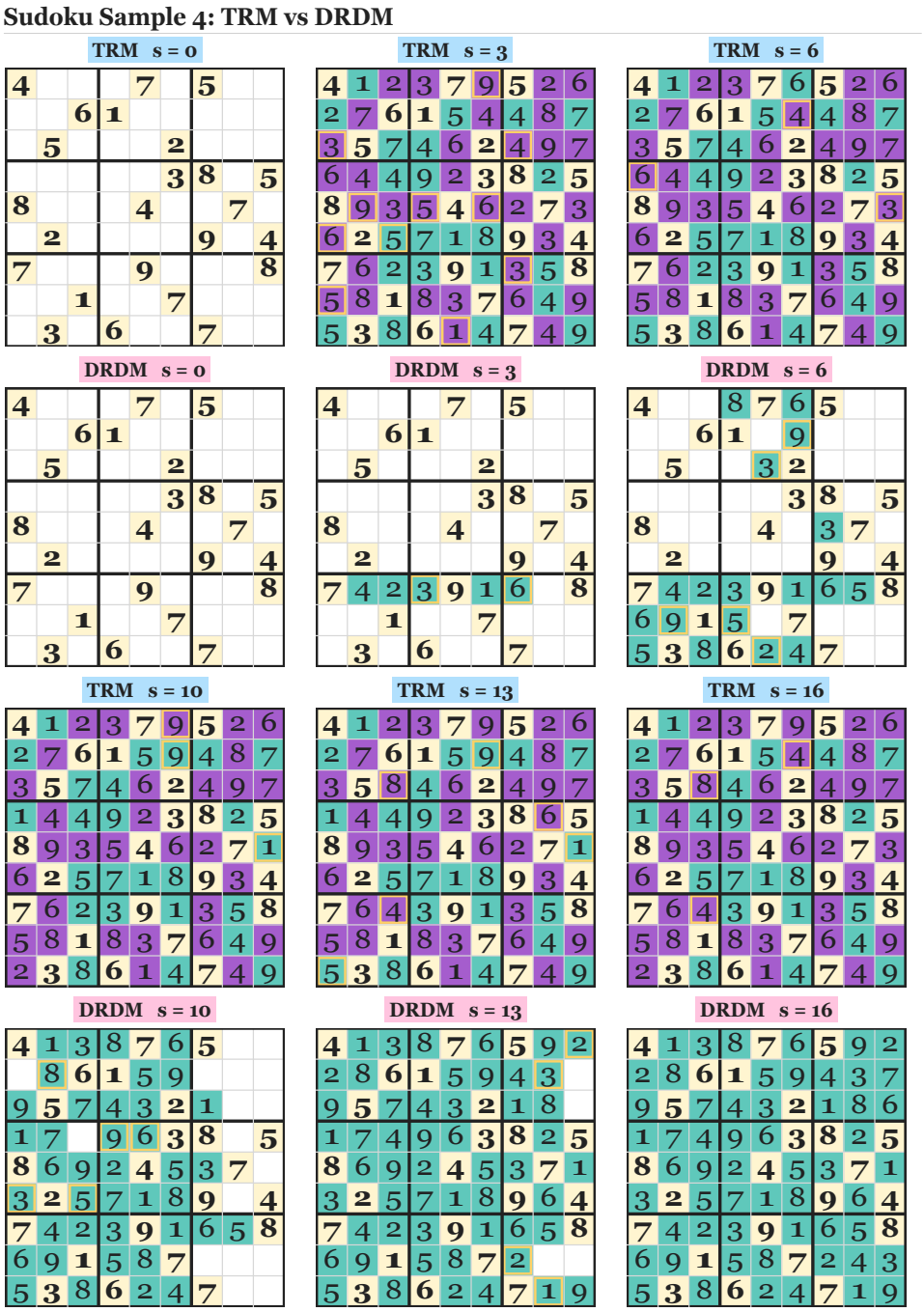


Figure 19: **Sudoku sample 4: from lower-left assignments to a correct grid.**

DRDM begins with a cluster of assignments along the seventh row, then fills much of the lower-left region by $s = 6$. At $s = 10$, the left side is densely populated while several right-side cells remain blank; those gaps are progressively filled through $s = 16$. TRM continues to display purple entries across multiple rows at its final shown stage, including the top row. This case pairs a fully mint DRDM completion with a dense candidate whose errors persist through successive revisions.

I.2 MAZE: DEVELOPING A CONNECTED PATH

The maze and endpoints remain fixed. Green cells mark predicted paths, with S and G indicating the source and goal. Consecutive stages show how the predicted path develops.

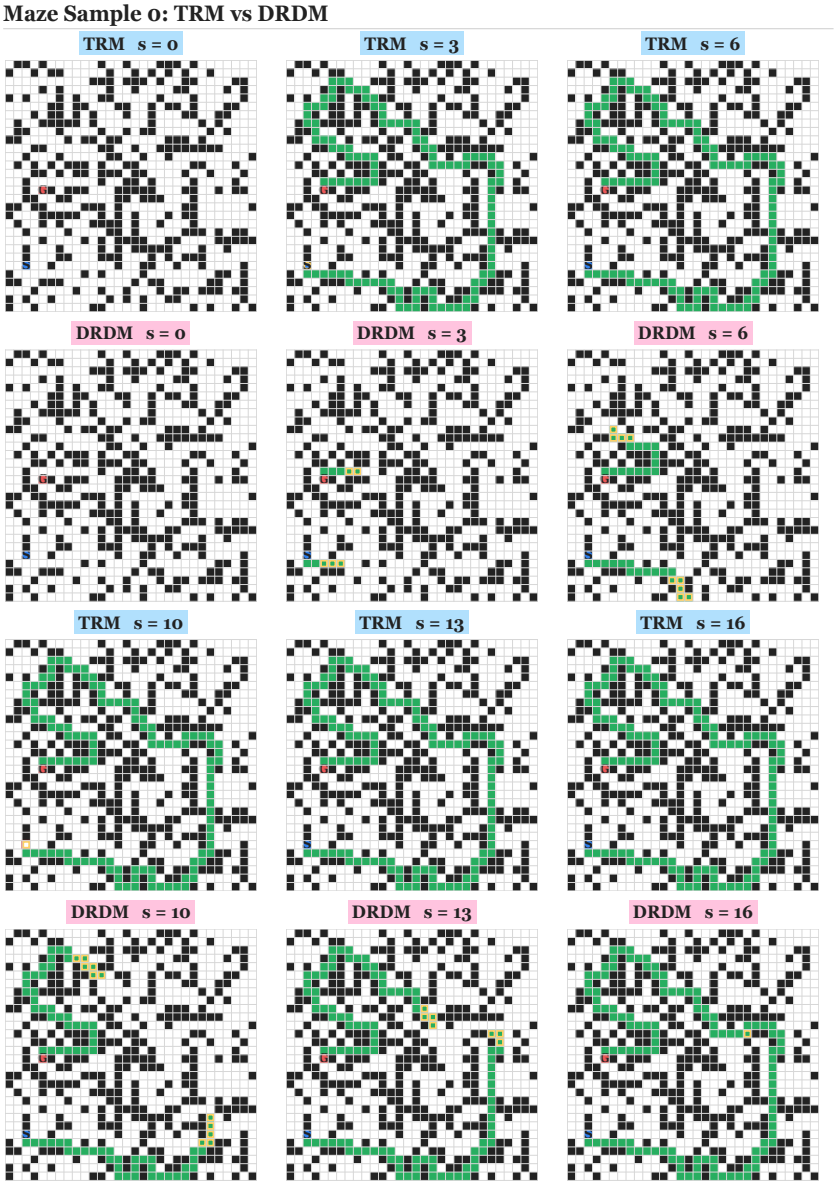


Figure 20: **Maze sample 0: joining through the upper-right passage.**

With S near the lower-left corner and G farther up the left side, DRDM grows two separate green sections at $s = 3$ and $s = 6$. By $s = 10$, one wraps around the upper-left passages while the other follows the bottom corridor toward the right boundary. The remaining separation shifts to the upper-right interior at $s = 13$, and the final panel bridges it. TRM displays the extensive winding route from $s = 3$, making the intermediate spatial separation particularly clear.

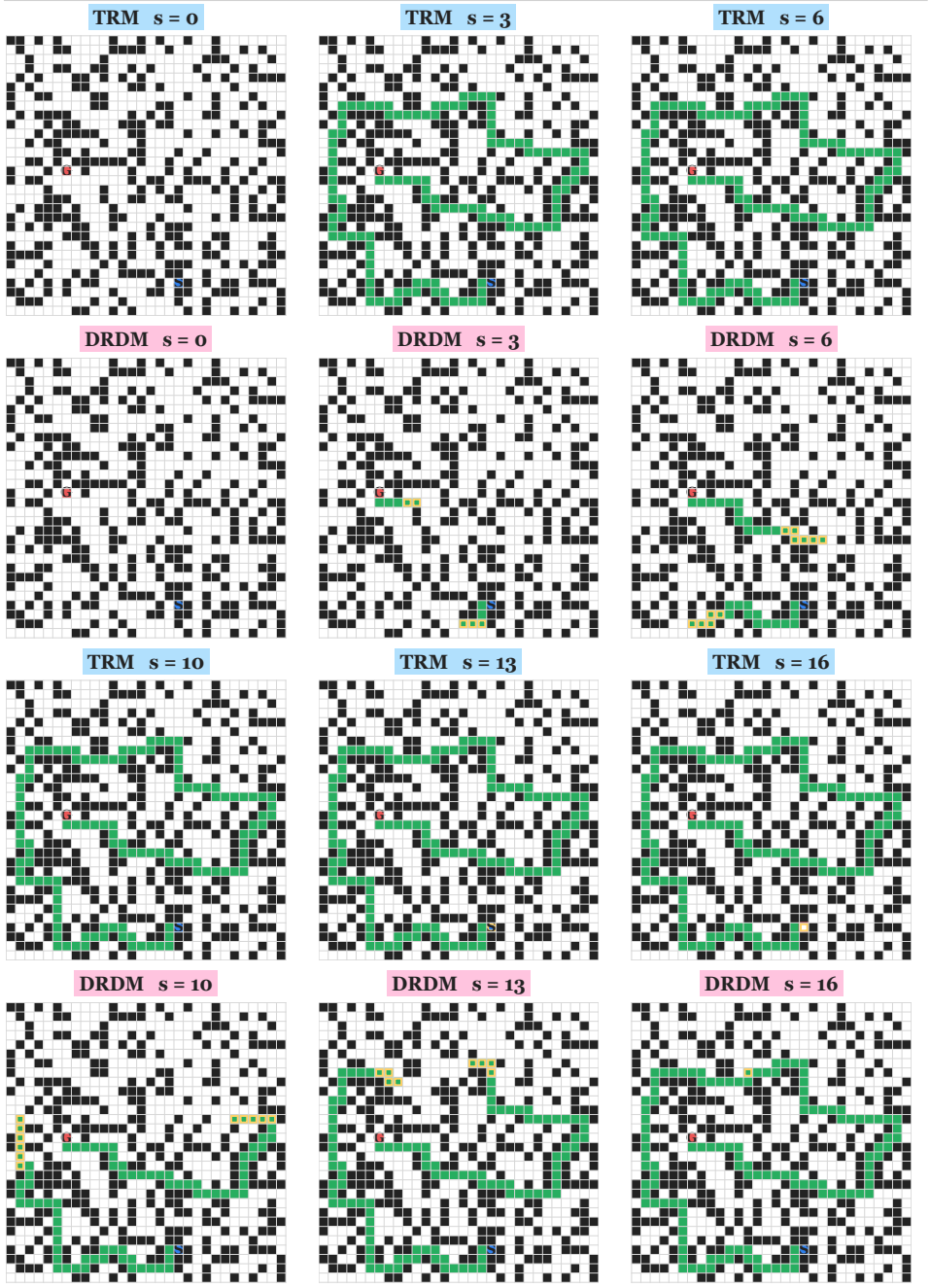


Figure 21: **Maze sample 1: closing the upper corridor.**

DRDM's early goal-side section extends rightward through the middle of the maze, while its source-side section develops along the bottom. At $s = 10$, the latter has turned upward along the left boundary and the former has wrapped around the right side. Their open ends occupy the upper part of the maze at $s = 13$; the final panel fills the intervening upper corridor. TRM already depicts this broad circuit through the maze at $s = 3$.

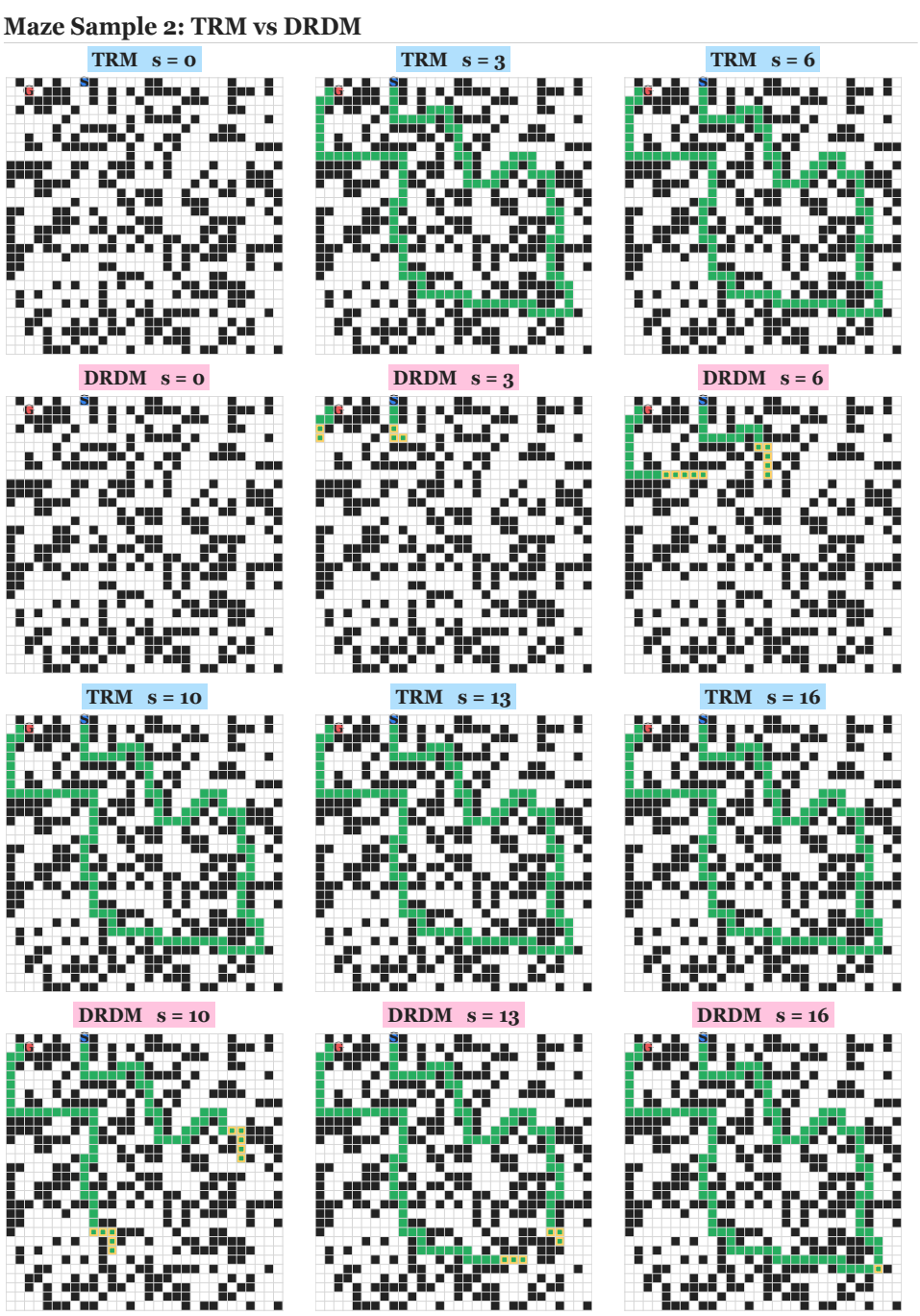


Figure 22: **Maze sample 2: a long detour between nearby endpoints.**

Although S and G lie close together near the upper-left edge, the displayed route descends deep into the maze. DRDM first marks short sections near both endpoints, then extends one down the left interior and the other across the upper middle toward the right side. At $s = 13$, their remaining separation lies near the lower-right turn, which is filled in the final panel. TRM's $s = 3$ prediction already displays the long detour through the lower corridors.

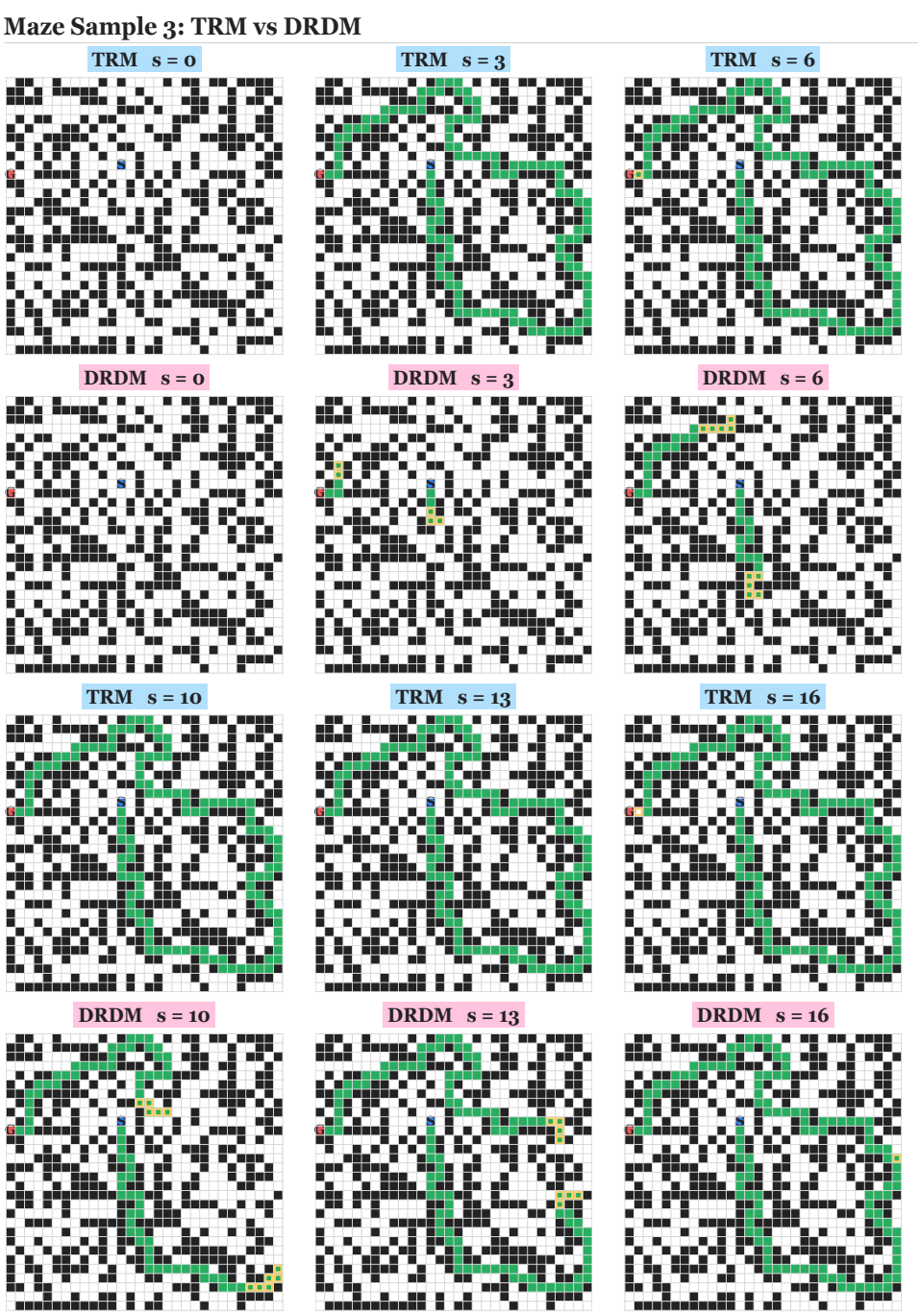


Figure 23: **Maze sample 3: connecting around the right boundary.**

S sits near the center-left and G on the left boundary at a similar height. DRDM develops the goal-side section upward and the source-side section downward, producing separated upper and lower arms by $s = 10$. At $s = 13$, these arms reach the right side but remain separated along the boundary; the final panel adds the connecting bend. TRM marks the broad route from $s = 3$, while DRDM's intermediate panels expose its upper and lower portions separately.

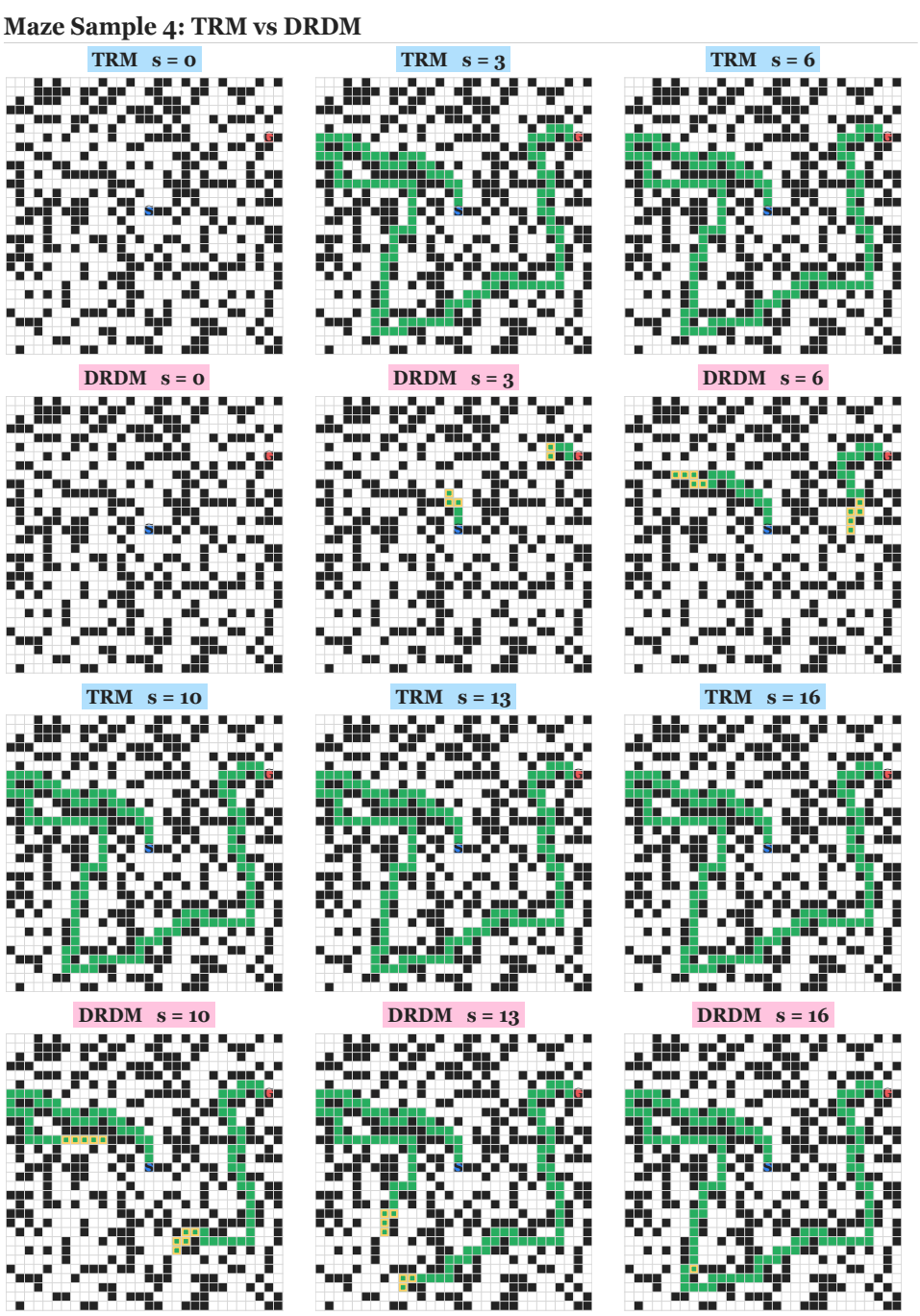


Figure 24: **Maze sample 4: completing the lower-left connection.**

Starting from central S and upper-right G, DRDM's green sections initially extend toward the upper-left passages and down the right side, respectively. By $s = 10$, the upper-left section has formed a broad hook, while the right-side section reaches the lower corridor. At $s = 13$, the remaining separation lies in the lower-left interior; the final panel fills the vertical connection there. TRM displays a similarly extensive upper-left and lower route throughout the shown prediction stages.

I.3 ARC: Refining Demonstration-Conditioned Transformations

Paired rows compare TRM and DRDM for each task. Inputs and targets precede stages 0, 3, 6, 9, 12, and 15, ordered from the upper to the lower block. The $s = 0$ panels repeat the input as a reference; subsequent panels show predictions conditioned on the demonstrations.

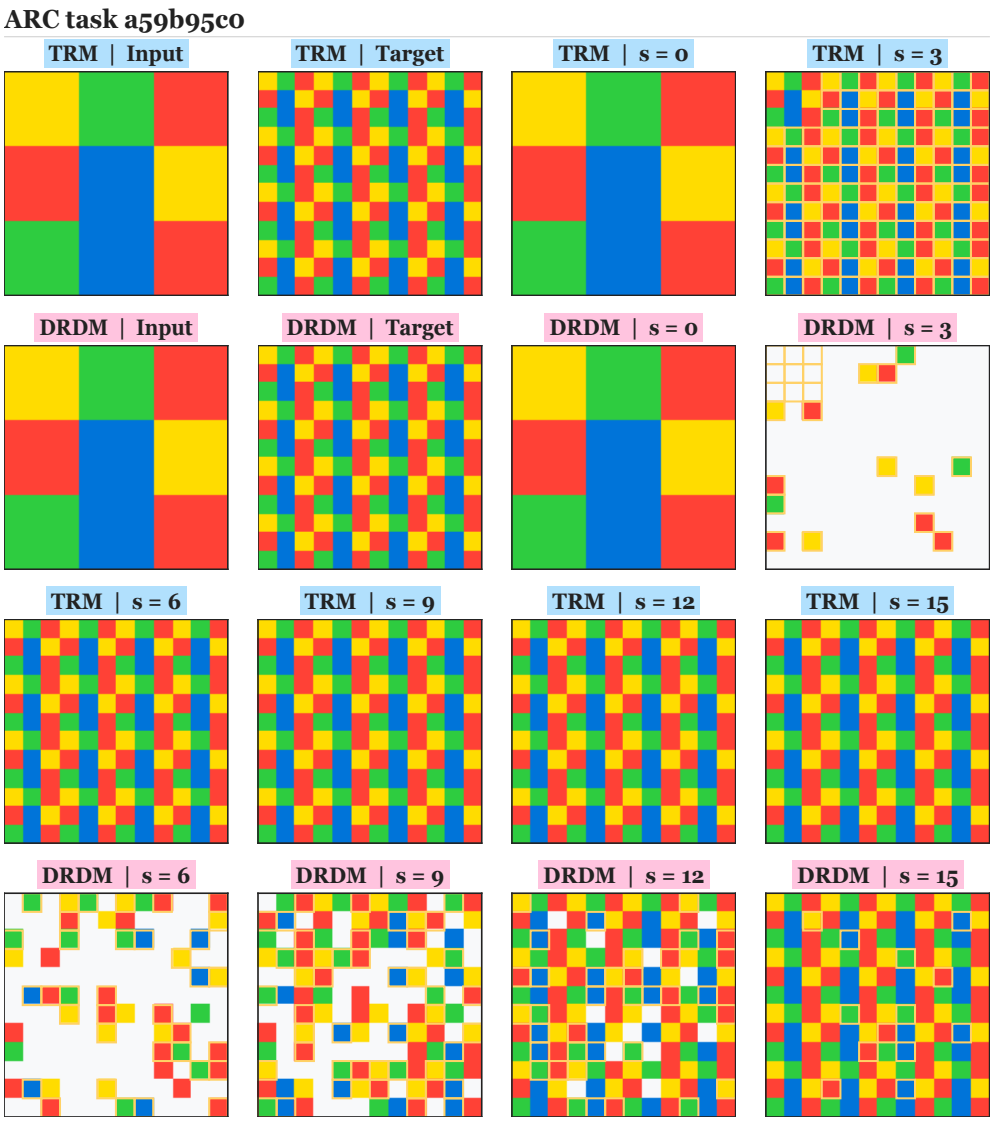


Figure 25: **ARC task `a59b95c0`: building a repeated color motif.**

The target tiles the input's three-by-three color arrangement across a larger grid, preserving the paired blue cells within each repeat. TRM displays a nearly complete periodic pattern at the first predicted stage. DRDM develops scattered colored cells into recognizable repeated patches between $s = 6$ and $s = 12$, then a continuous pattern at $s = 15$. The completed output aligns the local color motif across the repeated grid.

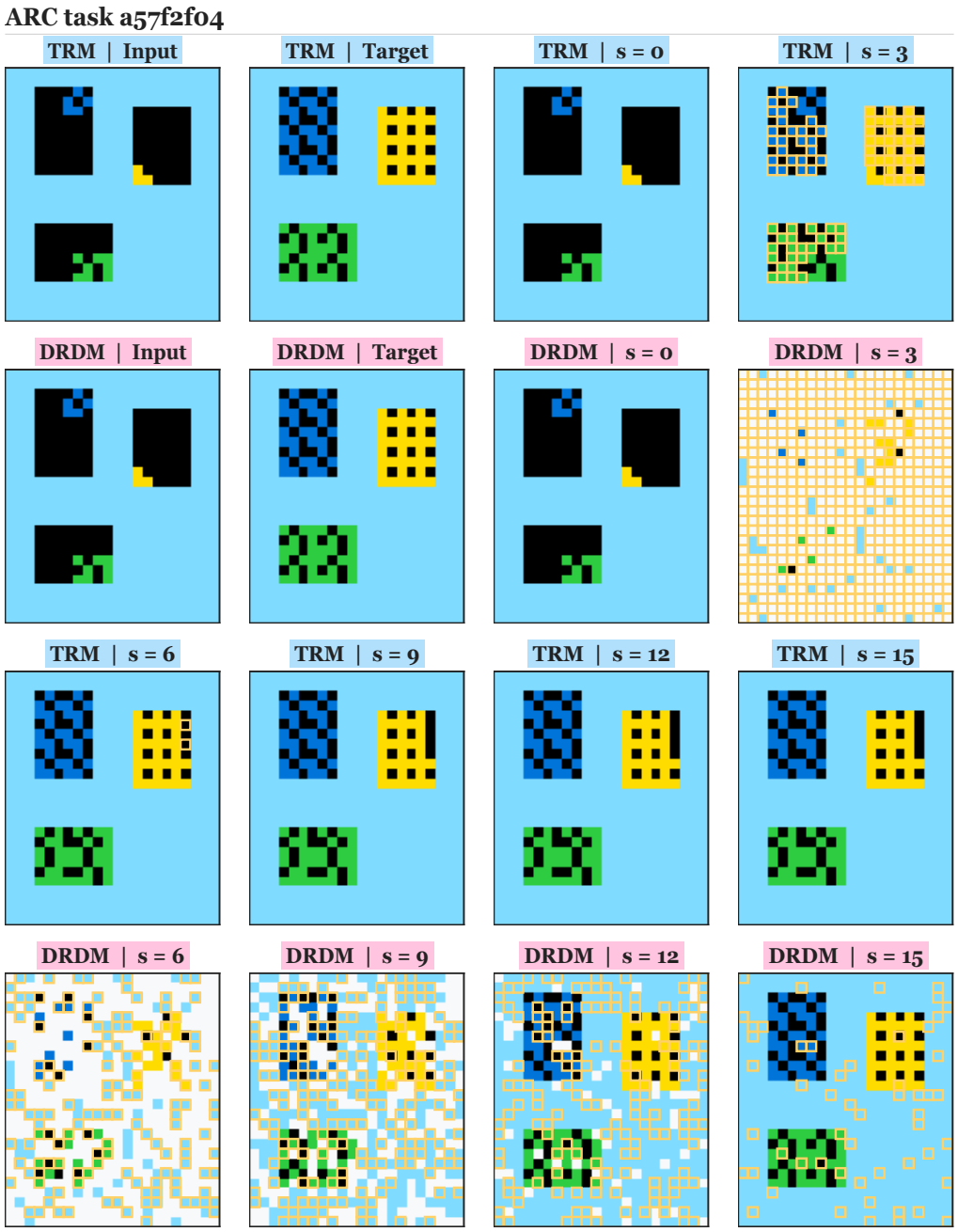
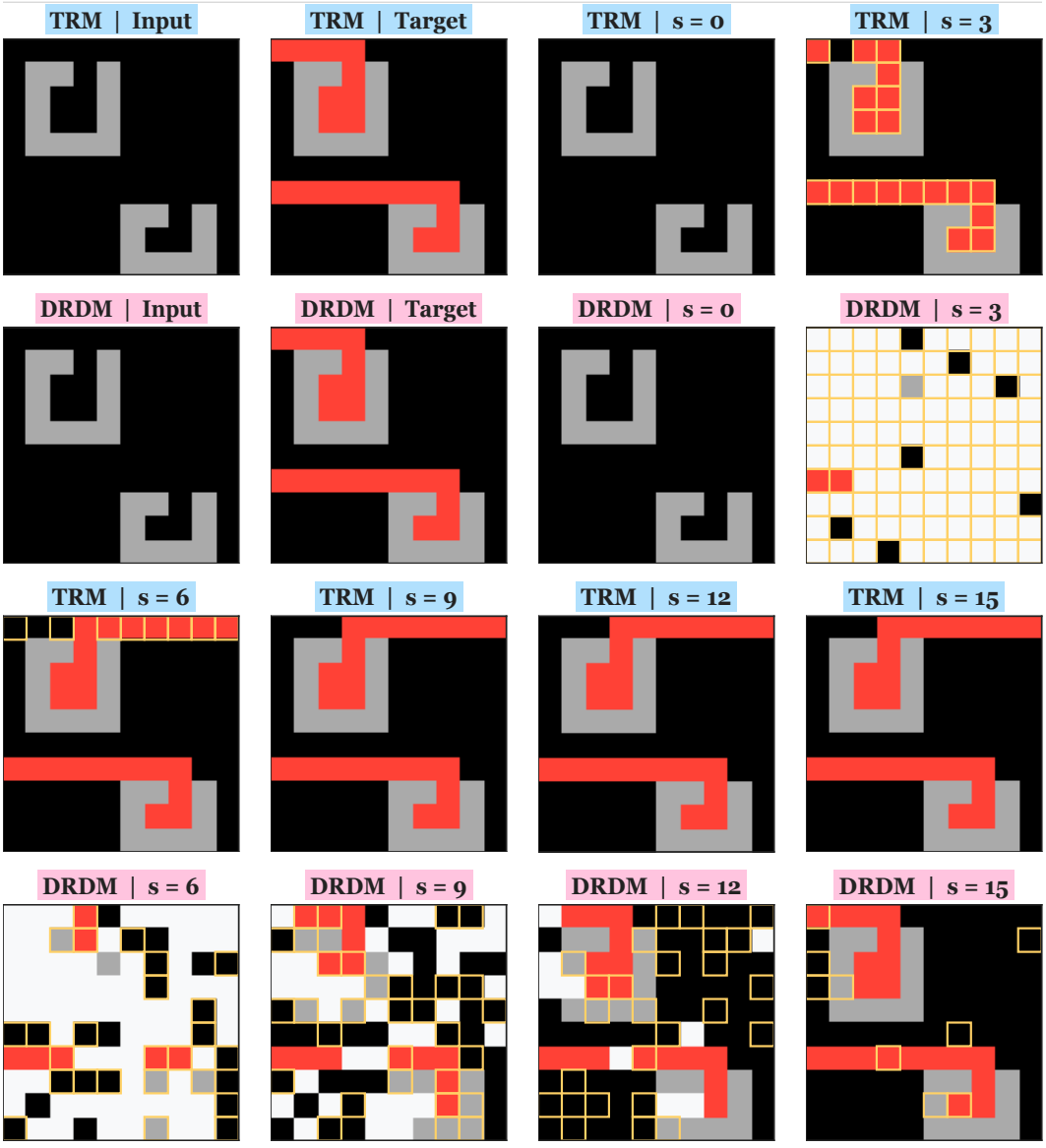


Figure 26: **ARC task `a57f2f04`: completing three object textures.**

The input provides small colored fragments within three black rectangles; the target extends each fragment into a distinct blue, yellow, or green texture. TRM establishes the object outlines early, with later changes concentrated inside the yellow and green rectangles. DRDM develops all three objects together as the cyan background spreads between them. By $s = 15$, their positions and color identities are distinct, while the internal black motifs remain coarser than the target, separating object placement from texture completion.



s = 0 repeats the input reference.

Figure 27: **ARC task `4e469f39`: recovering the direction of extension.**

The target inserts red paths into two gray U-shaped objects and extends both paths to the left boundary. TRM produces the lower connection early, but its upper extension points right from $s = 6$ onward. DRDM's intermediate red segments join into leftward connections at both heights as the gray objects and black background take shape. This case identifies a relational distinction beyond recognizing the objects: the horizontal continuation must follow the target's direction for each separate structure.

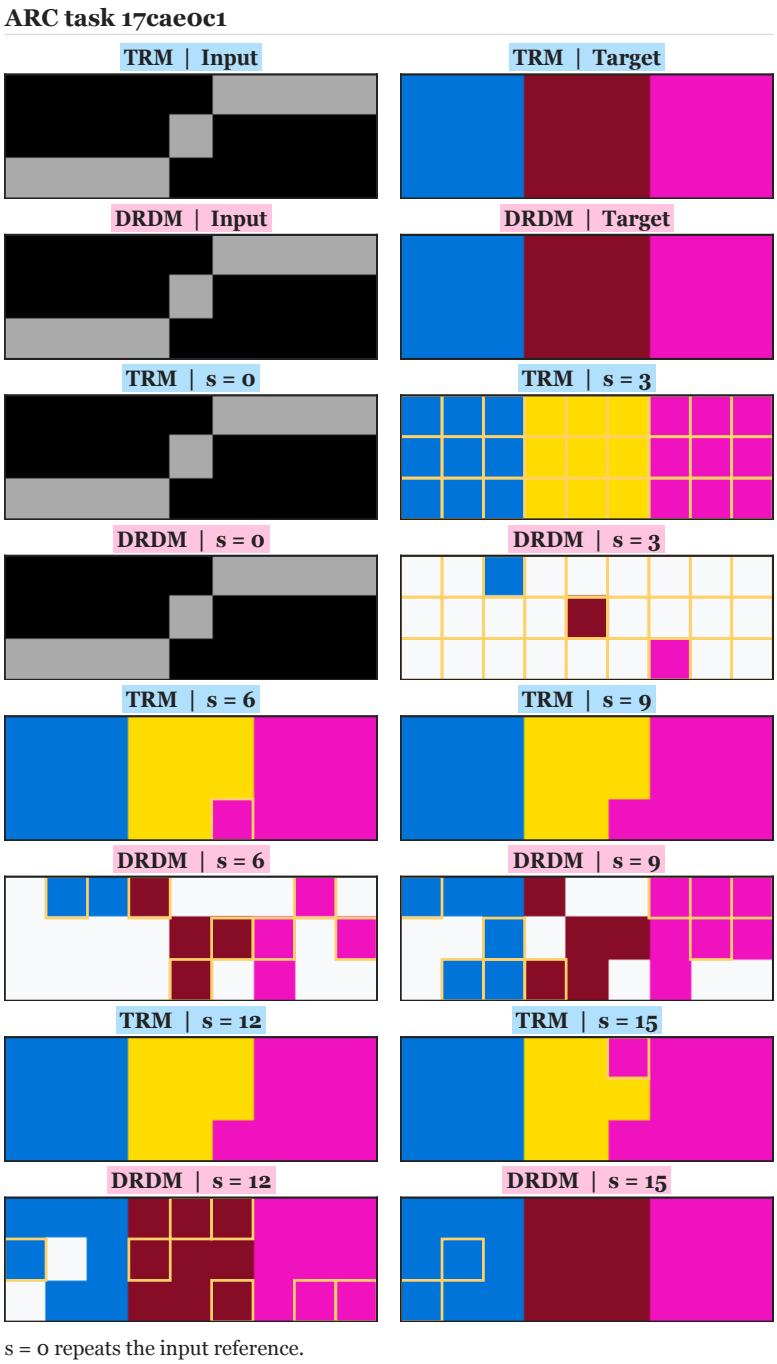
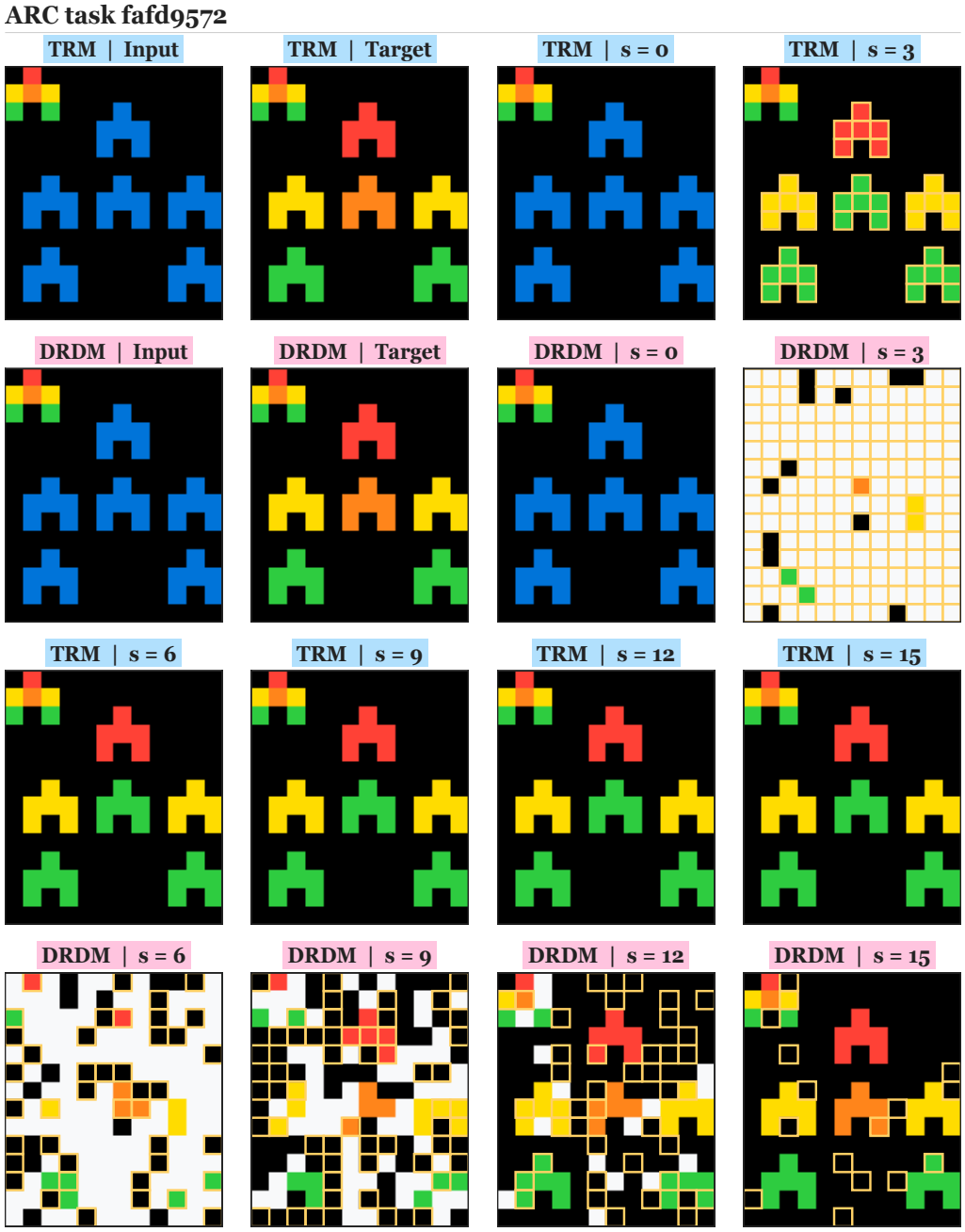


Figure 28: **ARC task `17cae0c1`: forming three uniform color bands.**

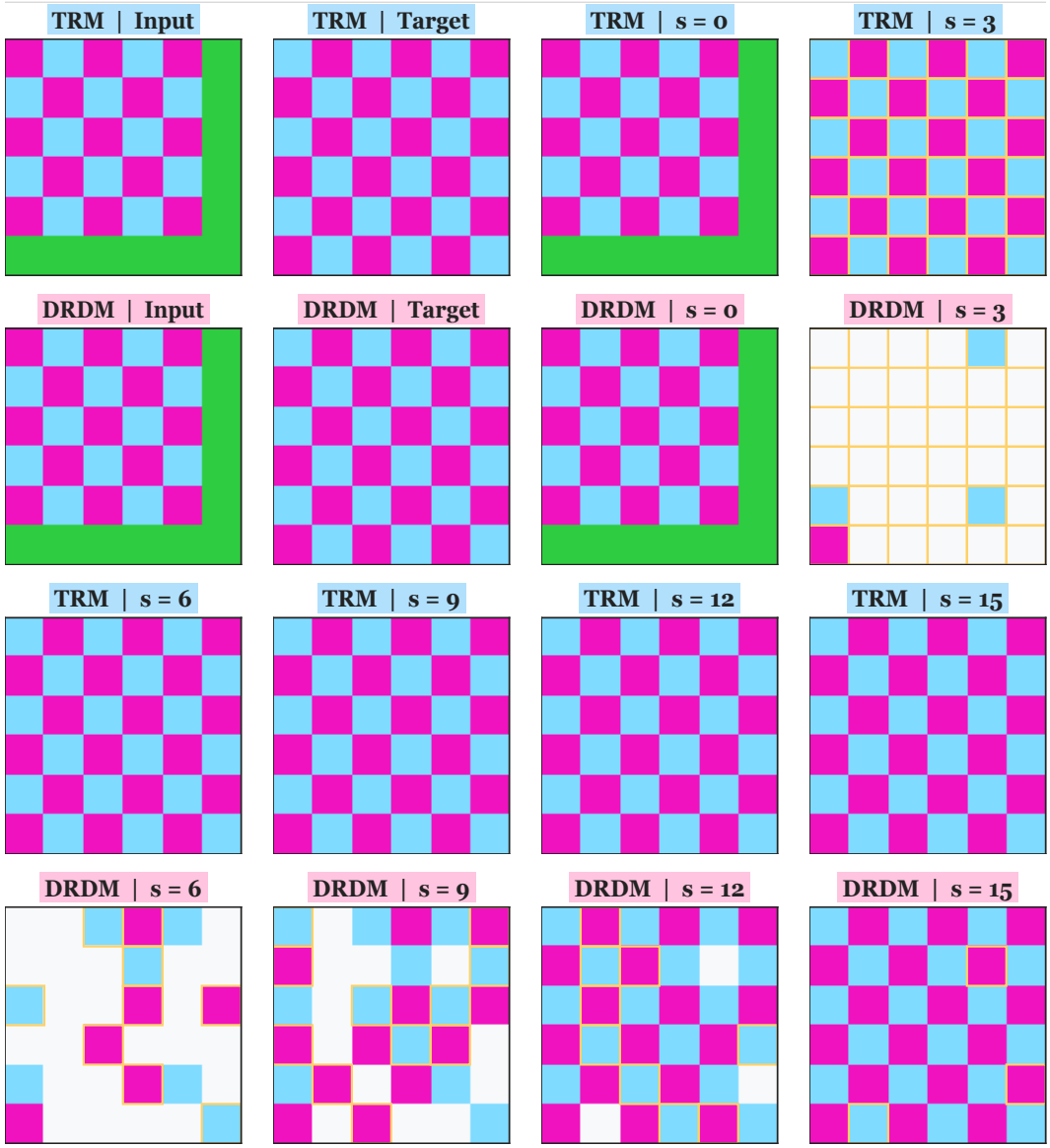
The stepped gray input is transformed into three equal-width vertical bands of blue, dark red, and magenta. DRDM's isolated early colors become broader regions by $s = 9$; the remaining interruptions at $s = 12$ disappear into uniform bands at $s = 15$. TRM forms the broad three-region arrangement immediately, but assigns yellow to the center and retains an irregular boundary with magenta. The comparison emphasizes that establishing a coarse partition and assigning its final colors are distinct parts of the transformation.



s = 0 repeats the input reference.

Figure 29: **ARC task `fafd9572`: assigning colors by object position.**

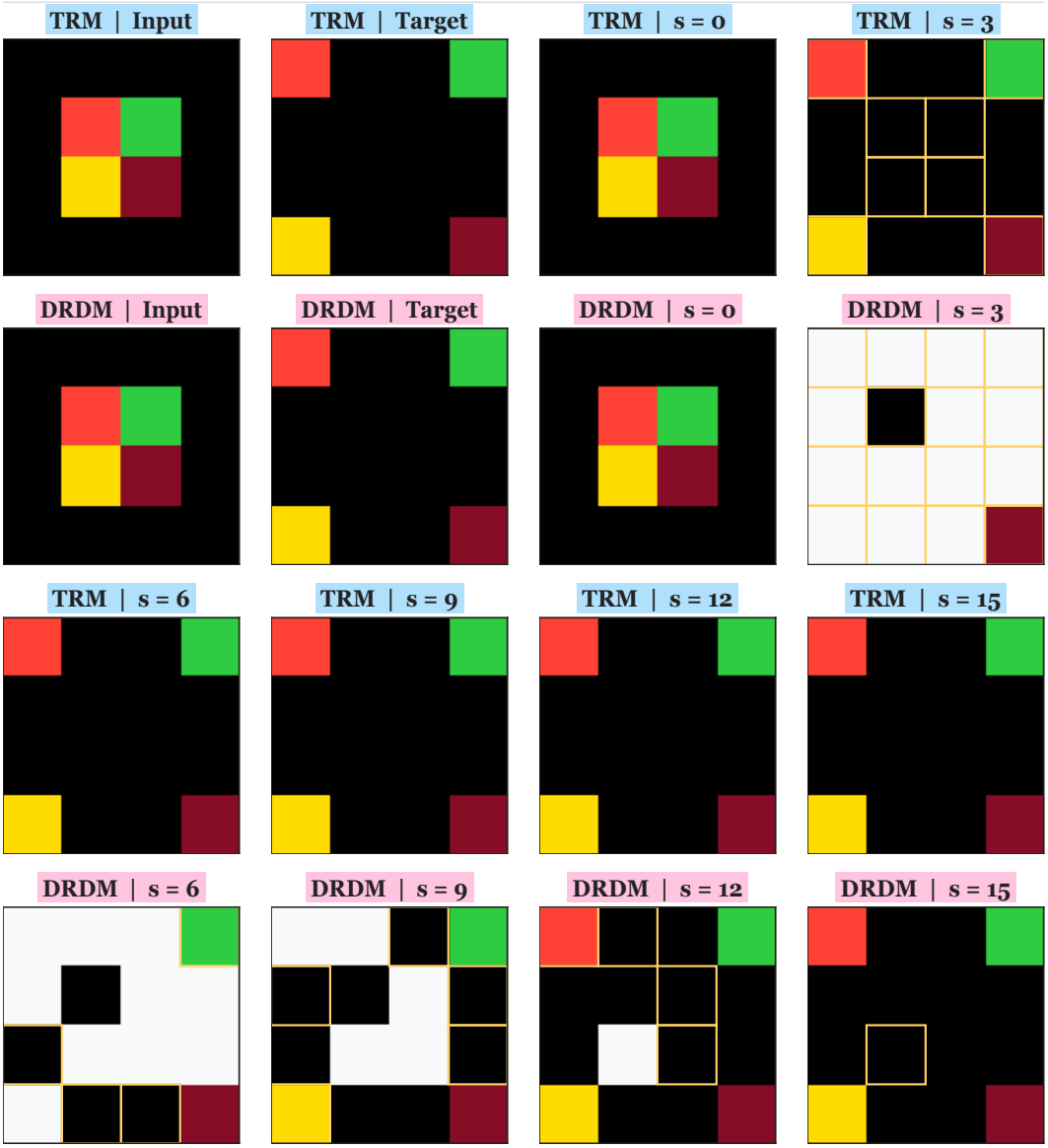
The six blue objects are recolored to reproduce the arrangement of the small multicolor example: red above, yellow on either side of an orange center, and green below. DRDM's later panels separate these roles as the object silhouettes emerge, with an orange central object visible by $s = 9$ and retained at $s = 15$. TRM preserves clean silhouettes early but colors the central object green throughout. The central object's identity therefore provides a specific test of transferring the example's spatial color arrangement.



s = 0 repeats the input reference.

Figure 30: **ARC task `50a16a69`: extending and shifting a checkerboard.**

The target replaces the green bottom and right borders with a full cyan-magenta checkerboard and reverses the input pattern's starting color. TRM displays this complete alternation at $s = 3$. DRDM develops the same arrangement through separated colored cells, larger alternating patches at $s = 9$, and a nearly complete grid at $s = 12$, reaching the continuous checkerboard at $s = 15$. The transformation combines spatial extension with consistent parity: adjacent cells must alternate, including those occupying the former green border.



s = 0 repeats the input reference.

Figure 31: **ARC task `66e6c45b`: moving a central block to the corners.**

The target moves the four colors from a central two-by-two block to the four corners, preserving their relative order around a black interior. TRM presents this arrangement from $s = 3$. DRDM first shows the lower-right dark-red corner, followed by the upper-right green corner at $s = 6$; the yellow and red corners are visible by $s = 9$ and $s = 12$, respectively. The remaining interior white cell becomes black at $s = 15$, completing the separation between corner colors and background.