# Probabilistic Tiny Recursive Model

**Amin Sghaier**
Mila – Quebec AI Institute
ILLS & ETS Montreal

**Ali Parviz**
Mila – Quebec AI Institute

**Alexia Jolicoeur-Martineau**
Independent

{amin.sghaier, ali.parviz}@mila.quebec
alexia.jolicoeur-martineau@mail.mcgill.ca

## Abstract

Tiny Recursive Models (TRM) solve complex reasoning tasks with a fraction of the parameters of modern large language models (LLMs) by iteratively refining a latent state and final answer. While powerful, their deterministic recursion can lead to convergence at suboptimal solutions, without escape mechanism. A common workaround relies on task-specific input perturbations at test time combined with answer aggregation via voting. We introduce **Probabilistic TRM (PTRM)**, a task-agnostic framework for test-time compute scaling that addresses this limitation through stochastic exploration. PTRM injects Gaussian noise at each deep recursion step, enabling parallel trajectories to explore diverse solution basins, and selects among them using the model's existing Q head (used for early stopping in the original TRM). Without requiring retraining or task-specific augmentations, PTRM enables substantial accuracy gains across benchmarks, including Sudoku-Extreme (87.4% to 98.75%) and on various puzzles from Pencil Puzzle Bench (62.6% to 91.2%). On the latter, PTRM achieves nearly double the accuracy of frontier LLMs (91.2% vs. 55.1%) at less than 0.0001x the cost, using only 7M parameters.

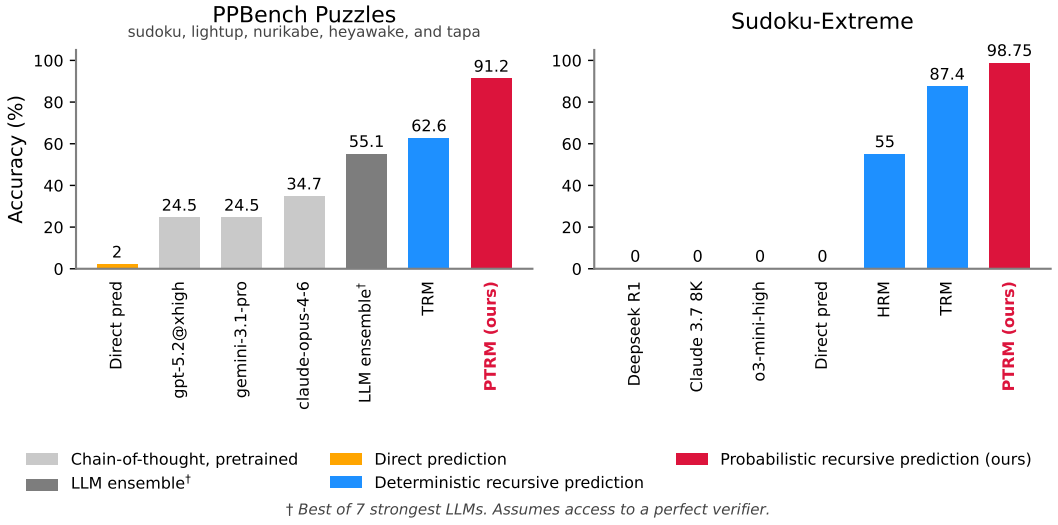


Figure 1: **PTRM performance comparison.** On various PPBench puzzles, PTRM boosts TRM performance by 28.6 points without any retraining. It outperforms the strongest single frontier LLMs by 56.5 points and an ensemble of the seven strongest LLMs (assuming a perfect verifier) by 36 points. On Sudoku-Extreme, PTRM reaches a state of the art 98.75%.

# 1 Introduction

Tiny Recursive Models (TRM) [1] achieve strong performance on complex reasoning puzzles with orders of magnitude fewer parameters than the large language models (LLMs) they outperform on tasks like Sudoku-Extreme [2] and ARC-AGI [3, 4]. TRM and its predecessor Hierarchical Reasoning Model (HRM) [2] represent an emerging architectural alternative to standard autoregressive reasoning models. Rather than autoregressively generating chains of token-level reasoning, they recursively refine a latent state. This approach produces a single deterministic answer per input, fitting well with tasks where the answer is unique.

Despite their strong performance, their deterministic inference does not make full use of their capabilities. We show that many of TRM's incorrect answers are from rollouts trapped in bad latent space basins (i.e., regions of the latent space which decode to incorrect answers and from which the deterministic recursions cannot escape). This observation, which aligns with recent mechanistic work on related models [5], suggests that TRM has the capabilities to solve significantly more problems but is limited by its standard inference procedure.

Although each puzzle has a unique correct answer, many distinct latent trajectories can reach it. This is analogous to reasoning LLMs, where many reasoning trajectories can lead to the same unique answer. However, being non-deterministic, LLMs can be randomly sampled in order to form different trajectories (including Chains of Thought and actual answer). By then selecting a trajectory using a voting mechanism or based on the answer's projected value (via a verifier), LLMs can leverage test-time compute to achieve very high accuracy [6]. We propose a way to achieve similar test-time scaling performance gains by sampling stochastic latent trajectories, each producing a deterministic decoded answer, and selecting among the answers using the model's own Q head.

TRM's Q head is trained jointly (as a correctness classifier) with the rest of the network and is conventionally used only at training time for adaptive computation (ACT) [7]. It carries valuable information that the standard inference procedure discards.

We propose **Probabilistic TRM (PTRM)**, a test-time compute scaling framework that introduces a new width scaling axis. At inference we run $K$ parallel rollouts per puzzle, each receiving Gaussian noise injected into the latent at every deep recursion step. The noise causes rollouts to follow different latent trajectories and settle in different basins. Among the resulting candidate answers, the Q head is used to select the one most likely to be correct. **PTRM requires no training changes and no task-specific test-time augmentation**, yet, as illustrated in Figure 1, delivers substantial accuracy gains across diverse reasoning benchmarks.

# 2 Background: Tiny Recursive Model

Tiny Recursive Model (TRM) is a single network that iteratively refines a predicted answer $y$ to a question $x$ through recursive updates of a reasoning latent $z$. Specifically, a single **latent recursion** consists of $n$ updates to the latent state $z$ followed by one update to the predicted answer $y$, all using the same two-layer network $f_\theta$: $z \leftarrow f_\theta(x + y + z)$ $n$ times, then $y \leftarrow f_\theta(y + z)$.

$f_\theta$ distinguishes the two update types by whether the input includes $x$. A **deep recursion** runs $T$ latent recursions in sequence, with only the final one retaining gradients, allowing the model to leverage a large effective depth while keeping training efficient.

Rather than doing one optimization step per sample, TRM is trained via deep supervision, which consists in keeping the previous latent state $z$ and answer $y$ as initialization (after being detached from the computational graph) for the next supervision step. This is done for up to $N_{sup}$ supervision steps. The loss at each step is calculated using cross entropy between the predicted answer logits $f_O(y)$ (where $f_O$ is a linear output head) and the ground truth $y_{true}$. This trains the network to progressively refine its prediction across reasoning steps. At inference, the recurrence can be unrolled for more steps than during training, providing a depth axis for test-time compute scaling (additional steps may correct otherwise-incorrect answers).

Without halting mechanism during training, each puzzle stays in the mini-batch for $N_{\text{sup}}$ supervision steps rather than being replaced after each one. To avoid wasting compute on already-solved samples, an Adaptive Computational Time (ACT) halting mechanism is used. This is done by adding a binary cross entropy loss between a halting logit $\hat{q} = f_Q(y)$ (where $f_Q$ is a linear Q head) and the binary exact

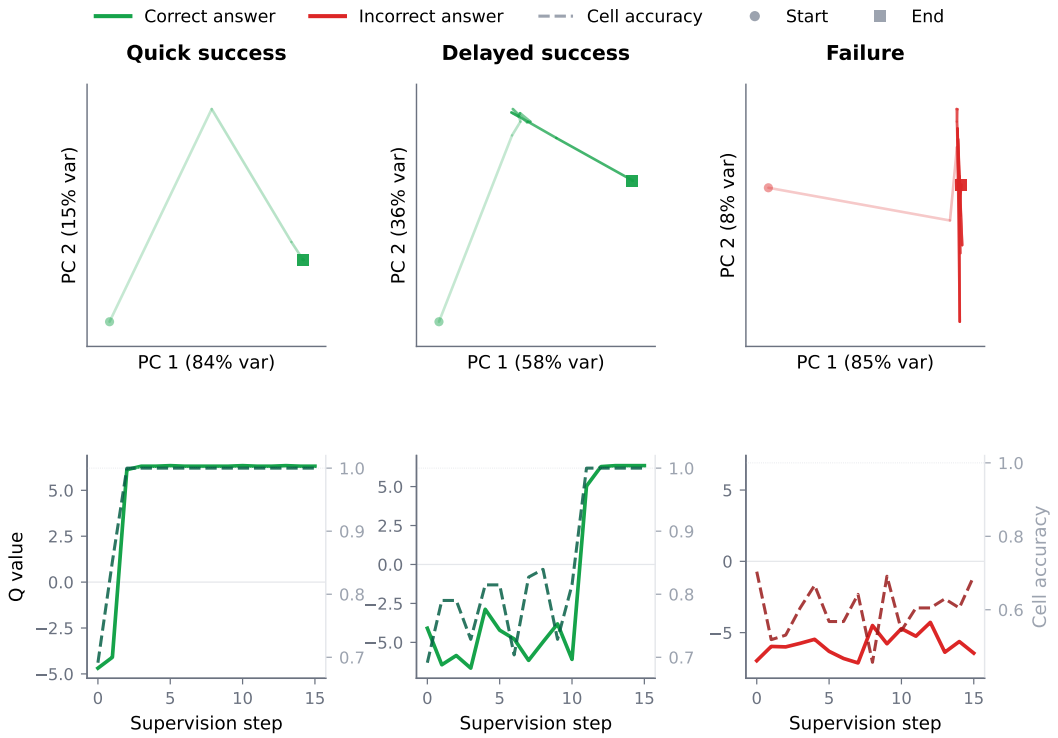


Figure 2: **TRM Trajectory Modes.** PCA projection of $y$ (top) and Q value (solid, left axis) with cell accuracy (dashed, right axis) across supervision steps (bottom) for three PPBench puzzles, illustrating three trajectory modes (left to right): quick success, delayed success, and failure (Sec. 3). Latents are projected into the principal plane per puzzle, so PC axes are not comparable across plots. Trajectories fade from light (early steps) to dark (later steps). Circle marks the start and square marks end.

correctness of the predicted answer $\hat{y} = \arg\max f_O(y)$: $\mathcal{L}_{\text{step}} = \text{CE}(f_O(y), y_{\text{true}}) + \text{BCE}(\hat{q}, \mathbf{1}[\hat{y} = y_{\text{true}}])$. The Q head thus allows the supervision loop to halt early on samples where $sigmoid(\hat{q}) > 0.5$, improving data efficiency. During inference, the Q head is not used, and the model performs $N_{sup}$ supervision steps to maximize answer correctness.

While TRM is powerful, it sometimes gets stuck into incorrect solutions. In the next section, we will investigate such failures cases in order to determine a way to remedy them.

## 3 Problem: When Does TRM Fail?

### 3.1 Analysis of failures and successes

We present observations about TRM that motivate our method. In this section, we train a TRM on multiple Pencil Puzzle Bench (PPBench) [8] puzzles and inspect the latent dynamics and Q head behavior across supervision steps on a held-out validation set. For each puzzle, we record the latent $y_t$ and the Q logit $\hat{q}_t = f_Q(y_t)$ at every supervision step $t = 1, \ldots, N_{\text{sup}}$, project the latents into the principal plane (PCA per puzzle), and jointly plot the Q value alongside cell accuracy (fraction of correct cells in the predicted answer) over supervision steps. Figure 2 shows paired PCA and Q/cell-accuracy plots for three representative puzzles, illustrating three trajectory modes we observe:

**Quick success**: the trajectory transitions in a few steps from its starting location to a convergence region and remains there. Cell accuracy and the Q value rise together and saturate near their maxima within the same few steps.

**Delayed success**: the trajectory initially oscillates around one region and remains there for multiple supervision steps before sharply escaping to a different region where it converges. During the initial

phase, the Q value is negative, and at the step where the trajectory escapes, both Q value and cell accuracy spike together.

**Failure**: the trajectory oscillates in a bounded region without converging. Cell accuracy never reaches near 100%, and the Q value stays negative for all supervision steps.

We refer to latent space regions that trajectories remain in across multiple supervision steps and exhibit similar cell accuracy throughout as basins. Basins where cell accuracy is near-maximal are good basins and basins where it is not are bad basins. Initially, failures and delayed successes behave similarly (both are caught in bad basins with negative Q). They diverge only later in their trajectories, when delayed successes find an escape to a good basin while failures remain stuck.

### 3.2 The Q head tracks trajectory quality

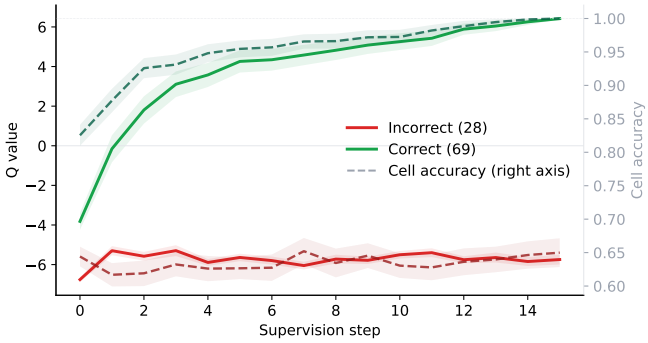


Figure 3: **Q value follows cell accuracy across reasoning.** Mean Q value (solid, left axis) and mean cell accuracy (dashed, right axis) over supervision steps, aggregated over 100 PPBench validation puzzles, separated by final correctness (green: correct, red: incorrect).

Across all three modes (failures, delayed successes, and quick successes), we find that the Q head's value closely tracks cell accuracy at every supervision step. To further confirm this, Figure 3 aggregates trajectories from 100 PPBench validation puzzles, separating them by final-answer correctness. The aggregate view corroborates the per-puzzle observation: mean Q and mean cell accuracy rise together on correct trajectories and remain mostly flat on incorrect ones. Moreover, at convergence, the Q logit sharply separates the two populations where $\hat{q} \approx +6$ ($sigmoid \approx 1$) for correct trajectories and $\hat{q} \approx -6$ ($sigmoid \approx 0$) for incorrect ones. The Q head is therefore a reliable learned indicator of whether a trajectory has reached a good basin.

Given that the Q head's ability to distinguish good from bad trajectories, a natural question follows: *can we leverage the Q head to identify better trajectories?* The main challenge is that the standard TRM is inherently deterministic, and thus cannot be used to sample different trajectories for a given problem. In the next section, we will show that by simply adding Gaussian noise to the latent state, we can sample different parallel trajectories and leverage the Q head to pick the best one.

## 4 Method: Test-Time Compute Scaling via Stochastic Rollouts

We propose **Probabilistic TRM (PTRM)**, an inference-time procedure that makes the TRM recursion stochastic and selects the best of $K$ resulting trajectories. PTRM requires no special training and can be readily applied to any pretrained TRM model. Furthermore it requires no task-specific augmentations. PTRM works as follows: at each supervision step, we add Gaussian noise (scaled by $\sigma$) to the latent state input. The Q head $f_Q$ scores each candidate latent output, and the one with the highest Q value is selected and then decoded using the model's output head $f_O$. The algorithm in Figure 4 (left) states this formally. PTRM offers two complementary benefits: 1) it enables trajectories to escape bad basins where deterministic TRM remains stuck, and 2) it introduces *width* as a new axis for test-time scaling.

### 4.1 Escaping bad basins

In Sec. 3, we found that some failed deterministic trajectories are caught in bad solution basins in latent space, with no way to escape. PTRM lets us test whether stochastic perturbations are enough for some of the rollouts of a previously failed puzzle to reach a good solution basin. Figure 5 shows K=100 independent rollouts, from the same failed puzzle used in Figure 2 (which fails at K=1),

**PTRM Inference**

**Input:** puzzle $x$, rollouts $K$,
supervision steps $D$, noise scale $\sigma$
**for** $k = 1, \ldots, K$ in parallel **do**
  Initialize $z_0^{(k)}, y_0^{(k)}$
  **for** $t = 1, \ldots, D$ **do**
    $z_{t-1}^{(k)} \mathrel{+}= \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2 I)$
    $z_t^{(k)}, y_t^{(k)} \leftarrow \texttt{rec}(x, z_{t-1}^{(k)}, y_{t-1}^{(k)})$
  **end for**
  $\hat{y}^{(k)} \leftarrow \arg\max f_O(y_D^{(k)})$
  $\hat{q}^{(k)} \leftarrow f_Q(y_D^{(k)})$
**end for**
**return** $\hat{y}^{(k^*)}$, $k^* = \arg\max_k \hat{q}^{(k)}$

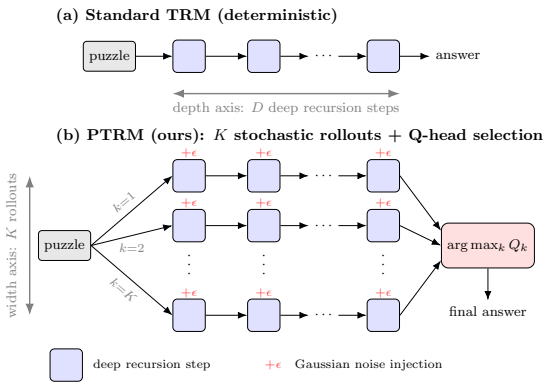


Figure 4: **Left:** PTRM inference procedure (the *rec()* function refers to a deep recursion step). **Right:** PTRM mechanism. (a) Standard TRM: a single deterministic rollout. (b) PTRM: $K$ stochastic latent rollouts with Gaussian noise $\epsilon$ at each deep recursion step, with the Q head selecting the final answer.

projected into the principal plane. Most rollouts (92%) remain stuck in the same bad basin, while a minority (8%) escape to a distinct region in latent space and produce correct answers. We also observe that recurrent noise creates a per-rollout probability of escape: at $K = 5$ no rollouts escape, at $K = 25$ one does, and at $K = 100$ eight do. This confirms that noise provides the stochasticity needed to occasionally find an escape trajectory.

### 4.2 Width scaling

Since more rollouts per puzzle compound the chance that at least one reaches a good basin, the number of rollouts $K$ is a natural quantity to scale. Given $K$ independent rollouts, pass@K (any rollout correct) is the oracle upper bound and best-Q@K (the rollout with highest $\hat{q}$ is correct) is a metric available at inference without a correctness oracle. The choice of Q as selector is motivated by Sec. 3's observation that Q accurately separates correct from incorrect trajectories (Figure 3).

Figure 6 shows pass@K and best-Q@K as $K$ grows, averaged over 3 seeds on the held-out PPBench validation set (sudoku, nurikabe, tapa, lightup, and heyawake). Both metrics rise from 76.4% at $K = 1$ to 89.5% at $K = 100$, a gain of 13 percentage points. Across all tested K, the gap between pass@K and best-Q@K stays under 1pp, making the Q head a strong verifier on this validation set. By contrast, mode@K (most frequent answer across rollouts) rises by only 1.3pp over the same range, showing that the width-scaling gains come mostly from the Q head's ability to identify correct solutions even when they are rare.

**Interaction with depth scaling.** Depth is another scaling axis already supported by TRM, which consists of running more deep recursions (supervision steps) at inference than the $N_{\text{sup}}$ the model was trained on. On the deterministic baseline ($K$=1), tripling the depth from 16 to 48 steps raises PPBench validation accuracy from 76.4% to 79.5% (+3.1pp). At higher $K$, depth scaling only provides additional gains on specific puzzle types such as sudoku (+4pp at $K = 100$). Both depth and width scaling can be seen as ways to explore the model's solution space. Since rollouts are independent and parallelizable while extra depth is sequential, width is the more practical scaling axis.

PTRM unlocks a simple and task-agnostic recipe for scaling TRM test-time compute. The next section evaluates the method across multiple benchmarks and against several baselines, including frontier LLMs.

## 5 Experiments

This section evaluates PTRM's performance on diverse reasoning benchmarks. We compare against the deterministic TRM baseline, a non-recursive direct-prediction baseline, and frontier LLMs. Across several PPBench puzzles [8], Sudoku-Extreme [2], Maze-Hard [2], and ARC-AGI 2 [4], PTRM substantially boosts the performance of each pretrained TRM using only inference compute.

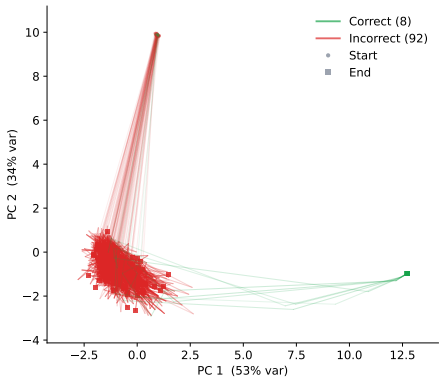


Figure 5: **Stochastic rollouts escape bad basins.** Principal plane projection of $K = 100$ independent rollouts of the same failed puzzle as in Figure 2 (right). 92 rollouts remain caught in the bad basin (red). 8 escape to a good basin and produce correct answers (green).

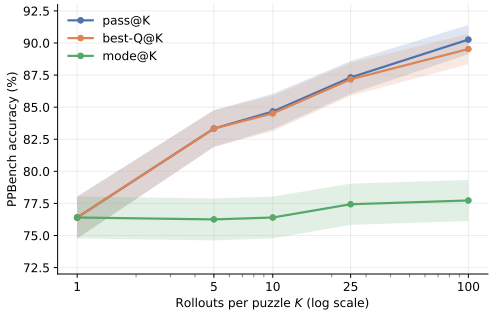


Figure 6: **Width scaling.** pass@$K$, best-Q@$K$, and mode@$K$ as $K$ grows, averaged over 3 seeds on a held-out PPBench validation set. The Q head is a strong verifier on the tested puzzles, consistently outperforming selection of the most frequent answer.

### 5.1 Setup

**Datasets.** Pencil Puzzle Bench (PPBench) [8] consists of 62,231 constraint-satisfaction pencil puzzles (from 94 puzzle types). From the full PPBench dataset, 300 puzzles (15 puzzles from 20 types) selected by Waugh [8] are held out to form the golden set. From the remainder we hold out a fixed-size validation set of 100 puzzles per puzzle type (50 for tapa, due to its smaller base size), and the rest forms the training set. We filter all three sets to puzzles of six types (sudoku, lightup, nurikabe, shakashaka, heyawake, and tapa) of grid size $9\times9$ for sudoku, and $10\times10$ for the rest. We use the validation set to track performance during training and select the final checkpoint. We report per-puzzle accuracy on five of these types on the golden set (TRM already reaches 100% on shakashaka, so we omit it from the reported results), with aggregate scores sample-weighted across types. We also report results on the *Sudoku-Extreme*, *Maze-Hard*, and *ARC-AGI 2* datasets.

**Models and inference.** For each benchmark we use a standard TRM checkpoint. For Sudoku-Extreme we use the TRM-MLP variant (which the TRM paper showed to be stronger on Sudoku), and for the other datasets, we use TRM-Att. PTRM inference uses $K$ parallel rollouts each running $D$ supervision steps with Gaussian noise of scale $\sigma$ added to the latent state at each supervision step. The selected configuration $(K, D, \sigma)$ varies by benchmark and is given alongside each result. Metrics are averaged across three seeds.

**Baselines.** To isolate the contribution of PTRM's stochastic rollouts from the underlying backbone, we report standard TRM performance (the same checkpoint as PTRM ran deterministically). For each dataset, we report the performance of frontier LLMs. For Sudoku-Extreme, Maze-Hard, and ARC2 we additionally report the published direct prediction and TRM baselines from [1].

**Cost estimation.** PPBench provides the dollar cost per attempt for each LLM. We convert PTRM's wall-clock to a comparable dollar figure using a single H100 at \$2.50/hr (standard cloud pricing [9]) so that cost $= \$2.50 \cdot t_{\text{puzzle}}/3600$, where $t_{\text{puzzle}}$ is the time (in seconds) to complete a puzzle.

### 5.2 Pencil Puzzle Bench

#### 5.2.1 Per-puzzle accuracy

Table 1 reports per-puzzle accuracy on the PPBench golden set. PTRM at $K{=}100, D{=}48, \sigma{=}0.2$ raises aggregate best-Q@$K$ from $62.6\%$ to $91.2\%$. Increasing supervision depth alone ($K{=}1, D{=}48$) gives a small boost over the standard TRM baseline ($K{=}1, D{=}16$). Most of the gain comes from scaling width (stochastic rollouts). The largest improvements are on puzzle types where

the deterministic baseline performed the worst (most headroom): sudoku improves from $46.7\%$ to $97.8\%$ and tapa from $40.0\%$ to $80.0\%$.

| % accuracy | # Params | sudoku | lightup | nurikabe | heyawake | tapa | **agg.** |
|---|---|---|---|---|---|---|---|
| Direct prediction | 27M | 0.0 | 0.0 | 0.0 | 14.3 | 0.0 | 2.0 |
| TRM ($K$=1, $D$=16) | 7M | 46.7 | 87.5 | 74.1 | **85.7** | 40.0 | 62.6 |
| TRM ($K$=1, $D$=48) | 7M | 57.8 | 87.5 | 74.1 | **85.7** | 40.0 | 66.0 |
| PTRM, best-Q@$K$ ($K$=100, $D$=16) | 7M | 93.3 | **100** | **88.9** | **85.7** | **80.0** | 89.8 |
| PTRM, best-Q@$K$ ($K$=100, $D$=48) | 7M | **97.8** | **100** | **88.9** | **85.7** | **80.0** | **91.2** |

Table 1: **PPBench per-puzzle accuracy on the golden set.** PTRM uses the same backbone as the deterministic TRM. Scaling depth alone ($K$=1, $D$=48) lifts aggregate accuracy by 3.4 points over the standard $D$=16 baseline. Combining depth with $K$=100 stochastic ($\sigma$=0.2) rollouts raises accuracy by 28.6 percentage points overall. The direct-prediction baseline is a larger transformer trained on the same data.

#### 5.2.2 Comparison with frontier LLMs on golden set

PPBench reported per-puzzle results for several frontier LLMs using two strategies: 1) direct response from a single prompt, and 2) multi-turn agentic strategy with verification. We report results for *direct* and *any* (best of any strategy attempted, including agentic). The agentic strategy gives the LLM substantially more resources than PTRM has access to. It provides the LLM the ability to iteratively verify each move with a perfect verifier. The direct strategy is the fairer comparison since, while it may use the model provider's reasoning harness, it does not have direct access to a multi-turn verifier (the LLM could still self-verify by writing verification code within the same response). We additionally observe that the agentic strategy was applied selectively in the published PPBench data: across the LLMs we compare against, only $9.6\%$ of direct failures on the golden set were retried with agentic. We restrict the comparison to the 7 strongest LLMs that attempted every puzzle in our golden set: `claude-opus-4-6@thinking`, `gpt-5.2@xhigh`, `gemini-3.1-pro`, `gpt-5.2@high`, `claude-sonnet-4-6@thinking`, `gpt-5.2@medium`, and `kimi-k2.5`. Table 2 lists the top 3 in each strategy block.

We additionally report an *ensemble* score formed from these 7 LLMs where a puzzle counts as solved if at least one of them solved it via any strategy. This ensemble setup is deliberately stacked against PTRM. It assumes a perfect verifier since, if any of the 7 LLMs produced a correct answer under any strategy, the ensemble counts it as solved, even though in practice we would not have access to an oracle verifier. Although it is not deployable, we include the ensemble to demonstrate that even under these heavily favorable conditions, frontier LLMs fall well short of PTRM. Ensemble cost-per-attempt averages over the attempts of all 7 models on each puzzle, and cost-per-correct divides total cost by the number of puzzles the ensemble solved.

Table 2 reports the comparison. PTRM exceeds the strongest single LLM (direct strategy) by 57 points aggregate ($91.2\%$ vs. $34.7\%$), and exceeds the LLM ensemble by 36 points ($91.2\%$ vs. $55.1\%$) despite the ensemble's stacked advantages. Cost per attempt is several orders of magnitude higher for LLMs than PTRM.

### 5.3 Sudoku-Extreme, Maze-Hard, and ARC-AGI-2

For each benchmark we use the standard TRM checkpoint trained as described in [1] without modification (TRM-MLP for Sudoku-Extreme and TRM-Att for Maze-Hard and ARC-AGI-2). Table 3 summarizes results on all three.

On Sudoku-Extreme, PTRM at $K$=100, $D$=64, $\sigma$=0.3 raises the deterministic baseline of $87.3\%$ to $99.06\%$ pass@$K$ and $98.75\%$ best-Q@$K$, achieving state of the art.

On Maze-Hard, PTRM at $K$=100, $D$=16, $\sigma$=1.0 reaches $95.63\%$ pass@$K$, an 11.83 point gain over the $83.8\%$ deterministic baseline. mode@$K$ gives the best PTRM accuracy here at $86.73\%$ ($+2.93$ points), with best-Q@$K$ slightly behind at $85.17\%$ ($+1.37$ points). While pass@$K$ shows that PTRM is able to unlock several correct answers, the Q head identifies them less reliably than on the previous benchmarks.

| % accuracy | sudoku | lightup | nurikabe | heyawake | tapa | **agg.** | **$/att.** | **$/corr.** |
|---|---|---|---|---|---|---|---|---|
| *Direct* | | | | | | | | |
| gemini-3.1-pro | 6.7 | 75.0 | 22.2 | 0.0 | 30.0 | 24.5 | $0.40 | $1.62 |
| gpt-5.2@xhigh | 20.0 | 50.0 | 0.0 | 0.0 | 50.0 | 24.5 | $1.79 | $7.29 |
| claude-opus-4-6@thinking | 0.0 | 87.5 | 44.4 | 0.0 | 60.0 | 34.7 | $2.91 | $8.40 |
| *Any strategy (direct or agentic)*† | | | | | | | | |
| gemini-3.1-pro | 6.7 | 87.5 | 33.3 | 0.0 | 40.0 | 30.6 | $10.38 | $33.91 |
| gpt-5.2@xhigh | 33.3 | 75.0 | 0.0 | 0.0 | 60.0 | 34.7 | $3.09 | $8.90 |
| claude-opus-4-6@thinking | 0.0 | 87.5 | 44.4 | 0.0 | 70.0 | 36.7 | $4.38 | $11.92 |
| *LLM ensemble*† | | | | | | | | |
| Any strategy (direct or agentic) | 46.7 | **100** | 44.4 | 0.0 | **80.0** | 55.1 | $2.66 | $38.51 |
| *Ours, trained from scratch, 7M parameters* | | | | | | | | |
| PTRM, best-Q@K | **97.8** | **100** | **88.9** | **85.7** | **80.0** | **91.2** | **$0.001** | **$0.001** |

Table 2: **PTRM vs. frontier LLMs on PPBench golden.** Per-puzzle accuracy and per-attempt / per-correct cost on the golden set. LLM costs are from PPBench. PTRM cost is estimated from H100 wall-clock (Sec. 5.1). The direct and agentic blocks list the 3 highest scoring LLMs on aggregate, and the ensemble row uses all 7 listed in Sec. 5.2.2. †Assumes access to a perfect verifier.

On ARC-AGI-2, the standard inference pipeline applies data augmentations and votes across them. PTRM adds $K$ stochastic rollouts per augmentation. For selection, we pick the rollout with the highest Q value within each augmentation, then vote across augmentations as in the standard pipeline. With $K$=25 and $\sigma$=0.2, PTRM lifts pass@1 from 7.36% to 8.47% and pass@100 from 14.31% to 15.97% over our deterministic TRM baseline, while matching it at pass@2.

| Method | # Params | Sudoku-Extreme Acc. (%) | Maze-Hard Acc. (%) | ARC-AGI-2 pass@1 | ARC-AGI-2 pass@2 | ARC-AGI-2 pass@100 |
|---|---|---|---|---|---|---|
| HRM | 27M | 55.0 | 74.5 | – | 5.0 | – |
| TRM | 5M / 7M† | 87.4 | 85.3 | – | 7.8 | – |
| *Ours* | | | | | | |
| Standard TRM, our reproduction | 5M / 7M† | 87.28 | 83.80 | 7.36 | **9.72** | 14.31 |
| PTRM | 5M / 7M† | **98.75** | **86.73** | **8.47** | **9.72** | **15.97** |

Table 3: **Sudoku-Extreme, Maze-Hard, and ARC-AGI-2 results.** For Sudoku-Extreme, $K$=100, $D$=64, $\sigma$=0.3. For Maze-Hard, $K$=100, $D$=16, $\sigma$=1.0. For ARC-AGI-2, $K$=25, $D$=16, $\sigma$=0.2. pass@$k$ for ARC-AGI-2 reports the top-$k$ predictions from the augmentation-voting pipeline. PTRM shows an accuracy improvement over standard TRM across all 3 benchmarks. †Following [1], 5M for Sudoku-Extreme (TRM-MLP), 7M for Maze-Hard and ARC-AGI-2 (TRM-Att).

### 5.4 Q head selection as $\sigma$ grows

With a higher $\sigma$ value, PTRM finds many correct solutions that the deterministic inference misses. For instance, on Maze-Hard, the deterministic model solves 83.8% of puzzles, but PTRM raises pass@$K$ to nearly 96%. The extent to which PTRM helps depends on the task, but on every dataset we tested, it unlocks correct solutions well beyond the deterministic model's reach.

TRM's jointly trained Q head serves as a strong verifier on most tasks. On PPBench and Sudoku-Extreme, best-Q@$K$ reaches values within a point of the saturated pass@$K$, so PTRM's exploration translates directly into accuracy gains. On Maze-Hard, more exploration (higher $\sigma$) produces significantly more correct rollouts, but the existing Q head is not able to identify them, leaving performance on the table. The gap between best-Q@$K$ and pass@$K$ represents headroom for a stronger verifier which is left for future work. Appendix B reports the full $\sigma$ sweep.

## 6 Related Work

A long line of work explores recursive computation for iterative reasoning and representation refinement. Early examples include Universal Transformers [10], Mixture-of-Recursions [11], Deep Thinking models [12, 13, 14], and HRM [2], all of which investigate the use of repeated computation steps to improve reasoning performance. More recent work has introduced methods to substantially accelerate TRM training [15], while TRM-style recursive architectures have also been extended to language modeling tasks [16].

Building on this broader perspective of recursive computation, a growing body of work studies latent-space reasoning through the reuse of hidden states. Hao et al. [17] propose continuous "thinking tokens" derived from Chain-of-Thought (CoT) traces [18], which are autoregressively generated and appended to the model context, enabling reasoning directly in latent space without producing intermediate textual outputs. Similarly, Zhu et al. [19] formalize learning by superposition and demonstrate improvements on tasks such as graph reachability. By avoiding explicit token sampling and implicitly representing multiple reasoning trajectories, these approaches may mitigate the unfaithfulness and backtracking often observed in standard autoregressive reasoning [20, 21].

Related to our work, Baek et al. [22] propose a generative version of TRM where the hidden state $z$ is sampled instead of deterministic. This improves performance on multiple tasks, but requires retraining. Efstathiou and Balwani [23] (concurrent work) propose a similar test-time compute method where they only apply noise in the initial hidden state $z$, while we apply noise at every supervision step. Furthermore, they test their method on a small subset of the Sudoku-Extreme dataset, and treat it as a proof-of-concept that needs to be developed and tested further. Note that Baek et al. [22] also tested applying noise to the initial $z$ with TRM and obtained negative results (no improvement in accuracy on two datasets).

Our observations in Sec. 3 are consistent with the mechanistic analysis of Ren and Liu [5], who identify spurious fixed points in HRM's latent dynamics on Sudoku-Extreme. Their method mitigates these attractors through a combination of task-specific training data augmentation, inference-time input perturbations, and model bootstrapping across training checkpoints, thereby effectively increasing test-time compute. However, these interventions are comparatively less general and less computationally efficient. In contrast, we observe analogous basin structure in TRM across multiple puzzle types and achieve attractor escape using a substantially simpler, task-agnostic mechanism: injecting Gaussian noise into the latent state at each supervision step while using a single deterministic checkpoint.

## 7 Conclusion

In this work, we introduced Probabilistic TRM (PTRM), a novel test-time scaling paradigm for Tiny Recursive Models (TRM) through parallel exploration and selection. This approach scales test-time compute using *width* ($K$ parallel rollouts), yielding substantially larger gains than *depth* scaling (increasing deep recursion steps) alone. PTRM requires no retraining and does not rely on task-specific data augmentations making it extremely easy to use and versatile.

By scaling both width and depth, PTRM obtains significant gains in accuracy when tested on a wide selection of puzzles. On PPBench (Sudoku, Lightup, Nurikabe, Heyawake, Tapa puzzles), PTRM nearly obtains twice the accuracy ($91.2\%$; $\$0.001$ cost) of ensemble of SOTA LLMs ($55.1\%$; $\$38.51$ cost) at less than $0.0001$x the cost. Furthermore, PTRM improves accuracy on Sudoku (from $87.4\%$ to $98.75\%$), Maze-Hard (from $83.80\%$ to $86.73\%$), and ARC-AGI (from $7.8\%$ to $8.47\%$ pass@1).

**Limitations.** Our experiments focus on reasoning puzzles rather than general tasks. We only test on a subset of PPBench puzzles. We are limited to puzzles with a small grid-size due to limited computational resources. It is not guaranteed that the method works as well for all types of problems (e.g., accuracy gains on ARC-AGI-2 and Heyawake are smaller).

**Future work.** It would be interesting to understand why some puzzles benefit from test-time scaling more than others. We suspect that problems that are harder to verify (e.g., ARC-AGI-2) benefit less from PTRM because the Q head may struggle to distinguish correct solutions from incorrect ones. Developing stronger verifiers than the existing Q head is an interesting direction for future work.

## References


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

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

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

[4] Francois 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.

[5] Zirui Ren and Ziming Liu. Are your reasoning models reasoning or guessing? a mechanistic analysis of hierarchical reasoning models. *arXiv preprint arXiv:2601.10679*, 2026.

[6] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

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

[8] Justin Waugh. Pencil puzzle bench: A benchmark for multi-step verifiable reasoning. *arXiv preprint arXiv:2603.02119*, 2026.

[9] Vast.ai. Rent h100 pcie gpus on vast.ai. `https://vast.ai/pricing/gpu/H100-PCIE`, 2026. Accessed: 2026-05-01.

[10] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[11] Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025.

[12] 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. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.

[13] 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. *Advances in Neural Information Processing Systems*, 35:20232–20242, 2022.

[14] Jay Bear, Adam Prugel-Bennett, and Jonathon Hare. Rethinking deep thinking: Stable learning of algorithms using lipschitz constraints. *Advances in Neural Information Processing Systems*, 37:97027–97052, 2024.

[15] Navid Hakimi. Form follows function: Recursive stem model. *arXiv preprint arXiv:2603.15641*, 2026.

[16] Yinxi Li, Jiaao Chen, Fang Wu, Jiakai Yu, Heli Qi, Weihao Xuan, Haokai Zhao, Pengyu Nie, Di Jin, and Xiangru Tang. Learning multi-step reasoning via persistent latent state propagation. In *Workshop on Latent {\&} Implicit Thinking {\textendash} Going Beyond CoT Reasoning*, 2026.

[17] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

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

[19] Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. *arXiv preprint arXiv:2505.12514*, 2025.

[20] Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

[21] Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, et al. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*, 2025.

[22] Junyeob Baek, Mingyu Jo, Minsu Kim, Yoshua Bengio, and Sungjin Ahn. Generative recursive reasoning models. *ICLR 2026 Workshop on AI with Recursive Self-Improvement*, 2026.

[23] Andreas Efstathiou and Aishwarya Balwani. Recursive reasoning as attractor landscape search: Mechanistic dynamics of the tiny recursive model. *Workshop on Latent & Implicit Thinking – Going Beyond CoT Reasoning*, 2026. URL `https://openreview.net/forum?id=kKps9W1K7n`.

# A Implementation Details

## A.1 Compute

We train and evaluate all models on a single NVIDIA H100 80GB GPU. PTRM introduces no additional training cost over standard TRM since it operates entirely at inference time.

## A.2 Models

All experiments use the standard TRM backbone [1] with the released architecture and training recipes. Following the TRM paper, we use the MLP variant (*TRM-MLP*, 5M parameters) for Sudoku-Extreme and the attention variant (*TRM-Att*, 7M parameters) for Maze-Hard, ARC-AGI-2, and PPBench. Layout and hyperparameters are unchanged from TRM.

## A.3 PPBench dataset construction

Sudoku-Extreme, Maze-Hard, and ARC-AGI-2 use the same checkpoints and data splits as TRM. The PPBench dataset is more recent and has previously been used only with frontier LLMs, so we detail how we built our training, validation, and golden splits.

**Source.** PPBench contains $62{,}231$ constraint-satisfaction pencil puzzles spanning 94 puzzle types. Of these, 300 puzzles (15 puzzles $\times$ 20 types) are held out as the *golden* benchmark set by Waugh [8].

**Filtering.** From the remaining $61{,}931$ puzzles we hold out a validation set by sampling 100 puzzles from each puzzle type (50 for `tapa`, due to its smaller base size), and the rest forms the training set. We then filter all three sets (training, validation, golden) to retain only puzzles of six types (`sudoku`, `lightup`, `nurikabe`, `shakashaka`, `heyawake`, `tapa`) at fixed grid sizes: $9\times9$ for sudoku and $10\times10$ for the others. Sudoku grids are padded with a pad token to $10\times10$, giving a uniform sequence length of seq_len $= 100$ across all six puzzle types. The deterministic TRM baseline reaches $100\%$ accuracy on shakashaka, so we exclude it from per-puzzle accuracy reporting (no headroom to compare against PTRM).

**Augmentation.** Each training puzzle is expanded into 10 examples using two augmentations: 1) *trajectory sampling*, where the input is set to a random intermediate solve state along the puzzle's solution trajectory rather than always the empty initial grid, while the label is always the fully solved grid; and 2) *dihedral transformation*, where a random dihedral transformation of a square grid, among the 8 possibilities given by 4 rotations $\times$ 2 {identity, reflection}, is applied to both the input and the label. For each puzzle, the first example is the unaugmented (initial state, solved) pair. The remaining 9 are randomly sampled (trajectory and dihedral transform). Validation and golden splits are not augmented.

**Resulting splits.** The merged multi-type splits use a unified vocabulary of 294 tokens and seq_len $=$ 100. Per-type sample counts are reported in Table 4.

| puzzle type | train | val | golden |
|---|---|---|---|
| sudoku | 7,810 | 97 | 15 |
| lightup | 9,504 | 65 | 8 |
| nurikabe | 15,180 | 55 | 9 |
| heyawake | 42,108 | 70 | 7 |
| tapa | 3,663 | 26 | 10 |
| shakashaka* | 20,702 | 62 | 12 |
| total | 98,967 | 375 | 61 |

Table 4: Per-puzzle-type sample counts in the PPBench splits used in training and evaluation. *Shakashaka is included in training but excluded from per-puzzle accuracy reporting because deterministic TRM already solves all evaluated shakashaka puzzles.

## B Noise Ablation

We ablate the inference noise level $\sigma$ on three benchmarks at $K{=}25$ ($K{=}100$ for Maze-Hard) and $D{=}16$ to keep the sweep tractable. For Sudoku-Extreme we randomly sample 1000 puzzles from the test set for the same reason. Figure 7 shows pass@$K$, best-Q@$K$, and mode@$K$ as a function of $\sigma$, averaged over three random seeds.

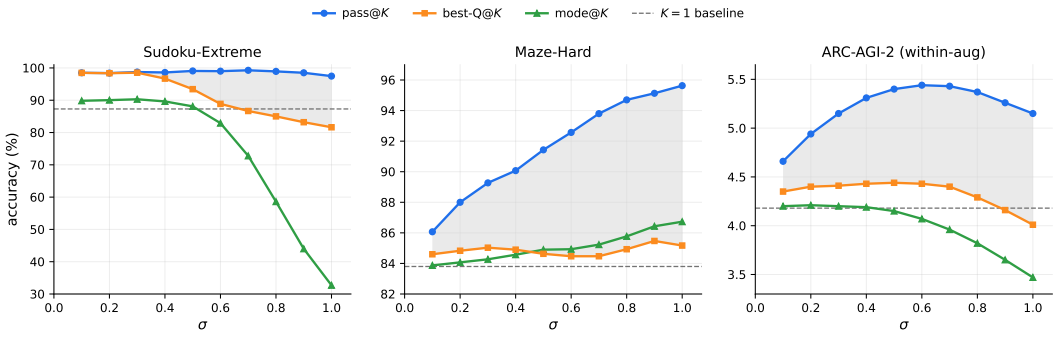


Figure 7: **pass@$K$, best-Q@$K$, and mode@$K$ across $\sigma$ per rollout batch.** On every task, increasing the inference noise consistently produces more correct rollouts (pass@K, blue) up to a task-dependent $\sigma$ value. The Q head (best-Q@$K$, orange) tracks the pass@$K$ ceiling closely on Sudoku-Extreme and leaves a larger gap on Maze-Hard and ARC-AGI-2. The shaded region represents the verifier headroom (accuracy that a better verifier could extract). mode@$K$ (green) has the edge over the Q head only on Maze-Hard. For ARC-AGI-2, metrics are per puzzle/augmentation to isolate the Q head's verification abilities from the augmentation pipeline.

On Maze-Hard pass@$K$ climbs from 83.8% (deterministic) to nearly 96% by $\sigma{\approx}1.0$ and then plateaus. On Sudoku-Extreme it is already near its ceiling at $\sigma{=}0.1$ and stays roughly flat across the sweep. On ARC-AGI-2 it peaks near $\sigma{=}0.6$ before declining. Q head selection nearly matches the ceiling (maximum pass@$K$) on Sudoku-Extreme while best-Q@$K$ peaks at $98.5\%$ (within a point of pass@$K$'s peak of $99.3\%$). On the other hand, the gap between best-Q@$K$ and maximum pass@$K$ is more pronounced on Maze-Hard and ARC-AGI-2 (headroom a stronger verifier could close).

## C Q-guided Langevin sampling

We initially explored Langevin sampling (using the Q head gradient) as a more principled exploration mechanism than the Gaussian noise injection used in PTRM. The idea is to better guide the stochastic search by additionally steering each rollout (using the Q head gradient) toward regions of high Q value. We ultimately found that the gain from this approach was entirely attributable to the Langevin noise term, with the gradient component contributing nothing measurable on top of the equivalent recurrent noise of Sec. 4. We document the approach here as a negative result.

**Motivation.** The Q head is trained as a correctness predictor over latent states. Let $f_Q(z)$ denote the head's scalar output. We treated $E(z) = -\log \mathrm{sigmoid}(f_Q(z))$ as an energy function over latent space. Empirical observations during early experiments suggested that regions of low $E$ correspond to good basins from which the decoded answer is likely correct. PCA visualizations of the latent dynamics showed that $\nabla_z f_Q$ points toward the good-basin region from both good-basin (correct) and bad-basin (incorrect) latents (Figure 8). This made $\nabla_z f_Q$ look like a valuable direction along which to push latents.

**Method.** We sample from the target distribution $p(z) \propto e^{-E(z)} = \mathrm{sigmoid}(f_Q(z))$ via Langevin dynamics where at the end of each deep recursion step $t = 1, \ldots, D$ we apply $N$ Langevin steps to the latent,

$$z \leftarrow z - \eta \nabla_z E(z) + \sqrt{2\eta}\,\xi, \quad \xi \sim \mathcal{N}(0, I),$$

The number of Langevin steps $N$ is the additional scaling axis under this scheme.

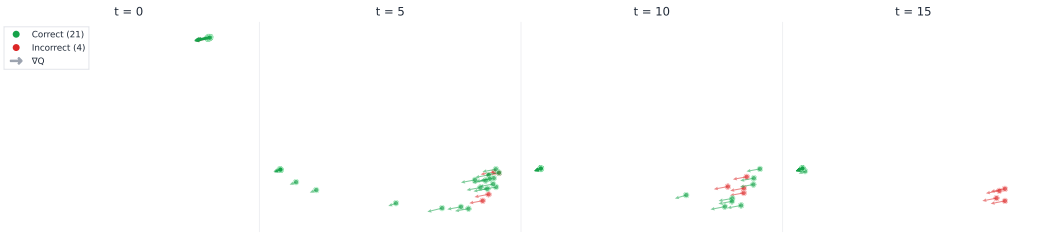


Figure 8: $y$ latents and their $\nabla_z f_Q$ gradients projected into the principal plane at several recursive/supervision steps, for multiple rollouts (using recurrent noise) of a single puzzle (correct rollouts in green, incorrect in red). Arrows are drawn at each latent in the direction of $\nabla_z f_Q$. From both good-basin and bad-basin latents, gradients point toward the good-basin region. This visualization motivated the Langevin sampling experiment described below.

**Tractable gradient computation.** TRM's original Q head is a linear projection on a single token, $f_Q(y) = w^\top y[:, 0] + b$, so its gradient with respect to this head's input is a constant vector independent of $z$. For $\nabla_z f_Q$ to be input-dependent, the gradient must flow back through the last latent recursion. This works but requires backpropagating through a full latent recursion at every Langevin step, which scales poorly with $N$. To make guidance tractable for large $N$, we replaced the linear Q head with an attention-pooled variant that reads the full latent and produces a scalar through a small nonlinear network. With this head, $\nabla_z f_Q$ can be computed by backpropagating through the head alone, which is $\sim 8\times$ faster per step and does not sacrifice accuracy.

**The gain came from the noise, not the gradient.** Comparing Langevin sampling against a noise-only ablation (with the same $\sqrt{2\eta}\,\xi$, but with the $-\eta\,\nabla_z E(z)$ term zeroed out) produced essentially identical accuracy at matched $N$. The gradient component contributed nothing measurable on top of the equivalent recurrent noise. This prompted us to focus on the noise-only formulation in Sec. 4, which is much more impactful since it is: 1) significantly simpler (no retraining, no test-time backpropagation), 2) applicable to any TRM checkpoint out of the box, and 3) equally effective.

## D Per-puzzle accuracy on the PPBench validation set

The main paper reports per-puzzle accuracy on the PPBench golden set (Table 1) for direct comparability with the LLM evaluations from Waugh [8] who used that set. For a lower-variance complement, Table 5 reports results on our validation set (313 puzzles across the five reported types vs. 49 for golden). Trends match the golden-set results: depth scaling alone ($K$=1, $D$=48) provides a small lift, and combining depth with stochastic rollouts ($K$=100, $D$=48, $\sigma$=0.2) raises aggregate best-Q@$K$ from 76.4% to 90.4%, a 14.0 percentage-point improvement. The biggest gains again are on puzzles where the deterministic baseline has the most headroom (tapa $\sim$ 40% to 71.8%, sudoku $\sim$ 69% to 93.3%). Types where the baseline is already near ceiling (heyawake at 96.7%) increase only marginally.

| % accuracy | # Params | sudoku | lightup | nurikabe | heyawake | tapa | **agg.** |
|---|---|---|---|---|---|---|---|
| Direct prediction | 27M | 0.0 | 10.0 | 4.0 | 14.0 | 0.0 | 6.2 |
| TRM ($K$=1, $D$=16) | 7M | 68.7 | 83.3 | 76.0 | 96.7 | 39.7 | 76.4 |
| TRM ($K$=1, $D$=48) | 7M | 74.0 | 84.0 | 76.7 | 98.0 | 41.0 | 78.3 |
| PTRM, best-Q@$K$ ($K$=100, $D$=48) | 7M | **93.3** | **93.3** | **84.7** | **100** | **71.8** | **90.4** |

Table 5: **PPBench per-puzzle accuracy on the validation set.** PTRM uses the same backbone as the deterministic TRM. Results on the larger validation set follow the same trends as on the golden set.