xRRMs: EXPLORATIVE RECURRENT REASONING MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent Reasoning Models (RRMs), including Hierarchical Reasoning Models (HRM) and Tiny Recursive Models (TRM), achieve strong performance on combinatorial reasoning tasks such as Sudoku while remaining parameter efficient. HRM and TRM are in principle looped transformers in which a shared module is applied repeatedly to refine a neural state. Conceptually, this computation is grounded in a fixed-point equation, using deep supervision and stop-gradients between iterations. Recently, stochastic variants of RRMs, such as Equilibrium Reasoners (EqR) and Probabilistic Tiny Recursive Models (PTRM), have been proposed. Beyond enabling generative modeling, stochasticity provides a natural mechanism for improving exploration of the solution space. In particular, these variants induce stochasticity by randomizing the initial latent states and by injecting noise during recurrent iterations. However, their gains rely on exploring multiple trajectories at inference, which raises the question of how to encourage exploration more directly. We address this with xRRM, a Recurrent Reasoning Model trained with strategies from Explorative Modeling: multiple sampled neural states are refined in parallel. At each deep supervision step, the loss is evaluated for all sampled trajectories, but gradients are computed only from the trajectory whose output is closest to the correct solution. This effectively implements a winner-takes-all objective and restricts backpropagation to the best-matching trajectory. On Sudoku-Extreme, xRRM achieves an average fully solved rate of more than 90% with a single evaluation candidate, improving over EqR by more than 20 percentage points. Test-time scaling with 32 evaluation candidates increases this value to more than 99%. We extend our approach to generative problem settings and evaluate it on the Game-of-24 benchmark, where multiple valid solutions per problem exist, investigating whether effective exploration extends beyond deterministic problem solving.

1 INTRODUCTION AND BACKGROUND

Recurrent Reasoning Models (RRMs), including Hierarchical Reasoning Models (HRM) (Wang et al., 2025), Tiny Recursive Models (TRM) (Jolicoeur-Martineau, 2025), Universal Reasoning Models (URM) (Gao et al., 2025), Symbol-Equivariant Recurrent Reasoning Models (SE-RRM) (Freinschlag et al., 2026), or Equilibrium Reasoners (EqR) (Huang et al., 2026) are neural networks that, akin to Recurrent Neural Networks (RNNs) (Schmidhuber, 2015), iteratively refine a neural state with a recurrent computation that is shared at each step. This can be viewed as iteratively approaching a state that satisfies an underlying fixed-point equation. The models are trained using deep supervision of the neural states: at each step, an output head maps the neural state to a prediction on which the loss is evaluated, while only the neural state is passed to the next step. To limit the computational cost of backpropagation, RRMs employ stop-gradient operations, so gradients are only computed with respect to the last computation steps.

The architecture design of RRMs follows a long tradition in recurrent architectures. Re-injecting a static input at every step and treating the resulting fixed point as the model output is a long-standing idea in recurrent networks (Almeida, 1987; Pineda, 1987; Schmidhuber, 2015). The recurrent computation itself has grown more expressive over time, with linear layers replaced by CNNs (Ernoult et al., 2019) and transformers (Dehghani et al., 2019).

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.

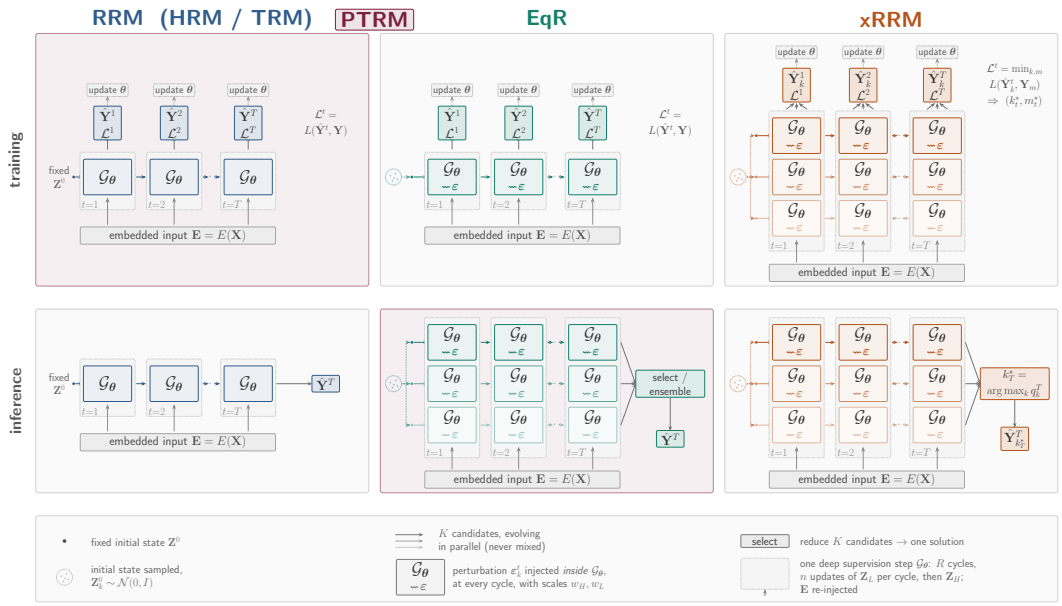


Figure 1: Schematic representation of different RRMs. **Left: HRM/TRM.** Both methods use a fixed initial state $\boldsymbol{Z}^0$ and are deterministic during training and inference. **Center: EqR** randomly samples $\boldsymbol{Z}^0$ and adds noise for each step. This allows ensembling for inference. **Right: xRRM (ours)** samples different initial states and maintains K trajectories during training to which winner-takes-all (WTA) loss is applied. This allows for more efficient exploration. At inference, the best trajectory based on the highest q-value is taken. **PTRM** is a mixture of those methods. It is trained as a deterministic TRM but adds noise during inference and evaluates with multiple candidates, similar to EqR.

Stochastic/Generative RRMs. Equilibrium Reasoners (EqR; Huang et al., 2026) introduce stochasticity into RRMs by randomizing the initial state and injecting noise at each reasoning step during training and inference. This enables test-time scaling by sampling multiple candidate solutions. Probabilistic Tiny Recursive Models (PTRM; Sghaier et al., 2026) build on a pretrained TRM and introduce stochasticity only at inference time, injecting Gaussian noise at each recursion step to enable stochastic exploration without retraining. In a similar direction, Generative Recursive Reasoning Models (GRAM; Baek et al., 2026) build on RRMs by formulating the reasoning process as a probabilistic latent-state model, enabling the sampling and exploration of multiple reasoning trajectories during inference.

Adaptive Computation. Instead of performing a fixed number of iterations, adaptive halting (Graves, 2016; Banino et al., 2021) limits computation time depending on task difficulty. HRM used adaptive computation time during training by introducing a halting head. PTRM found that such a halting head can additionally function as a trajectory evaluator.

Explorative Modeling (XM, Gladstone et al., 2026) is a training paradigm for generative models that enables end-to-end learning in settings where a single input can correspond to multiple valid outputs. In such ambiguous tasks, standard reconstruction losses tend to lead the model to collapse to a mean of outputs that does not match any of the actual modes well. The Forward XM procedure avoids this by drawing multiple candidate outputs for the same input and updating the model only through the candidate with the lowest loss. The authors also suggest a Reverse XM procedure, which compares the generated output against multiple ground truths and updates the model only through the target with the lowest loss, and thus targets precision.

Training a set of predictors so that only the best one is updated was introduced as multiple choice learning (Guzman-Rivera et al., 2012) and extended to enforce diversity (Guzman-Rivera et al., 2014) and to stochastic training of deep ensembles (Lee et al., 2016). The same principle underlies multiple-hypothesis prediction for ambiguous vision tasks (Rupprecht et al., 2017) and multimodal future prediction (Makansi et al., 2019). Perera et al. (2024) address the collapse and under-utilization of hypotheses through annealing.

Combinatorial reasoning problems are structured input-output problems in which a valid output must be constructed from discrete choices subject to global constraints. The input specifies the problem completely, and the validity of an output can be checked against the constraints without a reference solution. Classic solvers exploit this structure by exploring the space explicitly, e.g., by backtracking once a partial assignment violates a constraint. For such solvers, obtaining a valid output involves a sequence of dependent decisions rather than a single direct prediction. Problems of this type include Sudoku and Game-of-24. Some problems have a unique solution, such as Sudoku puzzles with sufficient clues, while others admit many valid solutions, such as Game-of-24. In both cases, several distinct trajectories through the space of partial solutions can lead to a valid output. These problems are difficult to solve correctly for plain LLMs (Hazra et al., 2024; Malek et al., 2025; Seely et al., 2025). Combinatorial reasoning problems with multiple valid solutions can be viewed as **conditional generative problems**. In addition to the general constraints governing the problem, the input specifies additional conditions that determine whether a given solution is valid. A generative approach should therefore not only be capable of finding a single valid solution but also of sampling from the entire set of valid solutions.

Problem Statement. In this work we are interested in applying RRMs to combinatorial reasoning problems. The deterministic dynamics of TRM and HRM may limit their ability to escape undesirable fixed points of the RRM attractor dynamics. EqR and PTRM, by contrast, obtain additional diversity by sampling multiple trajectories. We hypothesize that much of their exploratory advantage stems from this sampling, while the role of attractor dynamics in single-trajectory exploration has been less directly addressed. Since RRMs use deep supervision, which encourages latents to be decodable into complete solutions at every step, trajectories in alternate directions are penalized. For discrete reasoning problems, however, such deviations may be necessary to revise earlier decisions and explore alternative configurations.

Contribution and Suggested Solution. To improve exploration within the latent dynamics, we introduce **xRRM**, a recurrent reasoning model that maintains K parallel stochastic trajectories using learned perturbations during training. At each training step, only the candidate closest to a valid solution is selected to compute the task loss, leaving the remaining trajectories unconstrained by direct task supervision. In this way, xRRM keeps exploration in the latent state: multiple partial solutions can be refined in parallel without being serialized into an explicit reasoning trace. We show that this approach not only enables test-time scaling through trajectory sampling but also improves the exploratory behavior of individual inference-time trajectories. Fig. 1 contrasts xRRM with conventional RRMs, PTRM, and, EqR.

2 xRRMs: Explorative Recurrent Reasoning Models

2.1 Preliminaries: Notation and Exploration of RRMs

Recurrent Reasoning Models (RRMs). A recurrent reasoning model (RRM) maintains a state $\boldsymbol{Z}^t$ and updates it by repeatedly applying the same block $\mathcal{G}$. The input problem $\boldsymbol{X}$ is embedded once as $\boldsymbol{E} = E(\boldsymbol{X})$ and is available at every step, so that $\boldsymbol{Z}^t = \mathcal{G}(\boldsymbol{E}, \boldsymbol{Z}^{t-1})$ for $t = 1, \dots, T$. The initial $\boldsymbol{Z}^0$ differs between RRM variants. The block $\mathcal{G}$ typically consists of shared attention and shared MLP layers that are applied repeatedly within the block, but the exact realization differs between RRM variants. HRMs motivate a hierarchical design of $\mathcal{G}$, which has also been adopted in altered implementations by other variants such as TRM. In HRMs, the neural state $\boldsymbol{Z}^t$ is therefore decomposed into two parts, $\boldsymbol{Z}^t = (\boldsymbol{Z}_H^t, \boldsymbol{Z}_L^t)$. For each input $\boldsymbol{X}$, a target solution $\boldsymbol{Y}$ is required, which, in combinatorial reasoning tasks, can be obtained from a symbolic solver. At each recurrent step t, a prediction is obtained using an output head $\boldsymbol{O}$ as $\hat{\boldsymbol{Y}}^t = \boldsymbol{O}(\boldsymbol{Z}^t)$. The loss $L(\hat{\boldsymbol{Y}}^t, \boldsymbol{Y})$ provides deep supervision: gradients are propagated only through the most recent recurrent steps, with the state detached between steps. Equilibrium Reasoners keep this scheme but draw the initial state $\boldsymbol{Z}^0$ at random, compared to HRM and TRM, which learn $\boldsymbol{Z}^0$, and perturb the state during the update, i.e., $\boldsymbol{Z}^t = \mathcal{G}(\boldsymbol{E}, \boldsymbol{Z}^{t-1}, \boldsymbol{\epsilon}^t)$, where $\boldsymbol{\epsilon}^t$ is Gaussian noise scaled by a hyperparameter. PTRM follows in principle the same procedure during inference, but are trained in a deterministic way. A notation overview is provided in Section A.

Table 1: Each entry is the mean magnitude of the perturbation $\boldsymbol{\epsilon}_H \odot \boldsymbol{Z}_H \boldsymbol{W}_H$ and $\boldsymbol{\epsilon}_L \odot \boldsymbol{Z}_L \boldsymbol{W}_L$ applied to the latent states $\boldsymbol{Z}_H$ and $\boldsymbol{Z}_L$, respectively. The magnitudes are averaged over all positions and puzzles of the Sudoku-Extreme test set. The learned noise scale generally increases with the number of candidates.

Latent state	Candidates k					
	1	2	4	8	16	32
$\boldsymbol{Z}_H$	0.105	0.126	0.203	0.253	0.300	0.362
$\boldsymbol{Z}_L$	0.237	0.212	0.270	0.300	0.341	0.341

Adaptive halting. RRMs as introduced by HRM are typically evaluated with a fixed maximum number of deep supervision steps. During training, however, the number of steps can be determined adaptively using a learned halting head $\boldsymbol{f}_Q$. At deep supervision step t, the current recurrent state is mapped to a scalar halting score $q^t = \boldsymbol{f}_Q(\boldsymbol{Z}^t)$. The halting head is trained to predict whether the current state already represents a correct solution. If the halting score exceeds a predefined threshold, the recurrent computation terminates before reaching the maximum number of steps T. Otherwise, the current state is detached and passed to the next deep supervision step. Thus, adaptive halting allows the model to allocate different amounts of recurrent computation to problems of different difficulty.

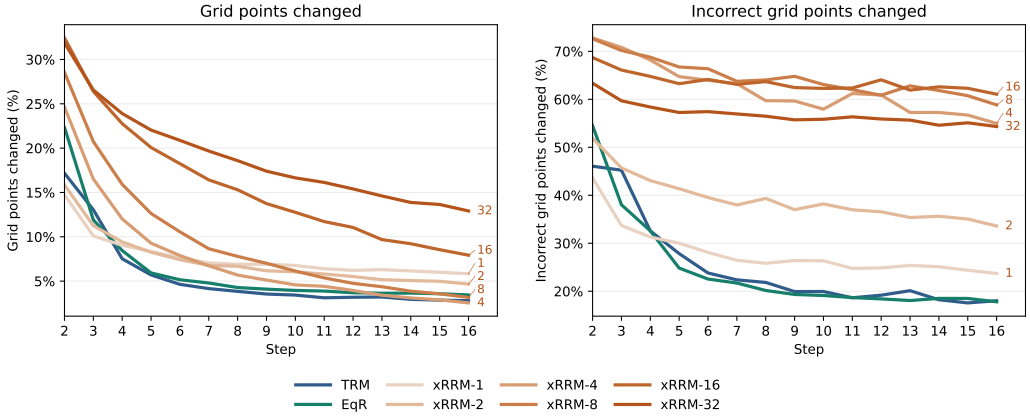


Figure 2: Changes in predicted digits relative to the preceding step during evaluation on Sudoku-Extreme across deep-supervision steps 2–16. (**Left**) Percentage of all digits whose predicted value changes between consecutive steps. (**Right**) Percentage of digits that were incorrect at the preceding step and whose predicted value changes at the next step. All models are evaluated with a single candidate during inference. The suffix of xRRM shows the number of candidates with which the model was trained. We see that incorrect grid points change more frequently, leading to increased exploration for yet imperfect predictions.

Exploration Behavior of Stochastic RRMs A motivation for introducing stochasticity to RRMs was better exploration. To get further insights on this, we studied EqR's exploratory behavior on a Sudoku task. We analyzed its iterative solution trajectories and compared them with those of TRM. As a measure of exploration, we tracked how often digits changed from one step to the next. A surprising observation is that EqR's trajectory appears almost as stable as TRM's (see TRM and EqR results in Fig. 2). This suggests that relatively little exploration may take place.

Potential Effects of Deep Supervision Training It is reasonable to expect that RRMs' use of deep supervision, which encourages neural states to remain decodable into complete solutions throughout the recurrent iterations, penalizes trajectories that deviate in alternative directions. For combinatorial problems, however, such deviations may be necessary to revise earlier decisions and explore alternative configurations.

2.2 EXPLORATIVE RRMS (XRRMS)

Explorative RRMs (**xRRM**) extend stochastic RRMs by incorporating the Explorative Modeling framework into their training. Although originally proposed for generative tasks, we approach the RRM exploration issue by leveraging the Explorative Modeling methodology for training RRMs. Rather than training a single recurrent trajectory per input, xRRM maintains multiple candidate trajectories and evolves them in parallel under a shared RRM instance. Stochasticity is used both at initialization and during the recurrent steps, where additional noise is injected with a learnable scale to encourage exploration. At each deep supervision step, all candidates produce a prediction, but only the one with the lowest task loss is selected for the backward pass, while the alternative trajectories are left free to evolve.

Remark. The Forward XM procedure in Gladstone et al. (2026) follows a winner-takes-all principle with respect to the candidates' losses (best-of-K) (Guzman-Rivera et al., 2012; Lee et al., 2016), i.e., the candidate attaining the minimum loss is selected. As noted in Gladstone et al. (2026), Forward XM generates multiple candidate outputs, with the candidate attaining the minimum loss contributing to the gradient for the parameter update. This observation can be formalized by the following lemma, assuming a unique minimizer.

Proposition 1 (**Winner-takes-all gradient under a unique minimizer**). *Let $\hat{\boldsymbol{Y}}_1, \ldots, \hat{\boldsymbol{Y}}_K$ denote K candidate solutions predicted by a function parameterized by $\boldsymbol{\theta}$, and let $\boldsymbol{Y}$ denote the ground truth. We define*

$$\mathcal{L}_{\mathrm{WTA}}(\boldsymbol{\theta}) = \min_{k \in \{1,\ldots,K\}} L\big(\hat{\boldsymbol{Y}}_k, \boldsymbol{Y}\big).$$

We assume that $L(\hat{\boldsymbol{Y}}_k, \boldsymbol{Y})$ is differentiable with respect to $\boldsymbol{\theta}$ for every k and that the minimizing candidate is unique:

$$k^\star = \operatorname*{arg\,min}_{k \in \{1,\ldots,K\}} \mathcal{L}\big(\hat{\boldsymbol{Y}}_k, \boldsymbol{Y}\big).$$

Then $\mathcal{L}_{\mathrm{WTA}}$ is differentiable at $\boldsymbol{\theta}$, and

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{WTA}}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} L\big(\hat{\boldsymbol{Y}}_{k^\star}, \boldsymbol{Y}\big).$$

Proof Sketch. Since the number of candidates is finite and the minimizer is unique, there exists a neighborhood of $\boldsymbol{\theta}$ in which the minimum is attained by the same candidate. Hence, the minimum operation locally reduces to the loss of the winning candidate, and its gradient is given by the gradient of that loss. □

Key Innovation. xRRMs extend RRMs by maintaining a pool of K candidate trajectories with independently initialized neural states during training, evolve them in parallel, and jointly train them using a winner-takes-all-based objective.

Architecture and Training. Every candidate k carries its own state and receives an independent perturbation at every step:

$$\boldsymbol{Z}_k^t = \mathcal{G}\big(\boldsymbol{E},\, \boldsymbol{Z}_k^{t-1},\, \boldsymbol{\epsilon}_k^t\big), \qquad \boldsymbol{\epsilon}_k^t, \boldsymbol{Z}_k^0 \sim \mathcal{N}(0, I), \qquad \hat{\boldsymbol{Y}}_k^t = \boldsymbol{O}\big(\boldsymbol{Z}_k^t\big), \tag{1}$$

At every step, all candidates are evaluated, but only the candidate prediction closest to the solution $\boldsymbol{Y}$ influences the overall loss. This means:

$$\mathcal{L}^t = \min_{k \in \{1,\ldots,K\}} L\Big(\hat{\boldsymbol{Y}}_k^t,\, \boldsymbol{Y}\Big) \tag{2}$$

As follows from Proposition 1, for unique minima only the best-matching candidate contributes to the parameter update, while the remaining candidates stay free to represent other modes of the output distribution. Computing the full product ($K \times M$) is expensive for large M, and an approximation is given in Algorithm 2 (Section B).

Extension to the generative case. For the case of combinatorial reasoning problems with multiple valid solutions $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_M$, we adopt the strategy of the Reverse XM procedure suggested by Gladstone et al. (2026) and minimizes the loss jointly over candidates and valid solutions:

$$\mathcal{L}^t = \min_{k \in \{1,\ldots,K\},\, m \in \{1,\ldots,M\}} L\left(\hat{\boldsymbol{Y}}_k^t,\, \boldsymbol{Y}_m\right).$$

Changed Halting Mechanism. In addition to the task loss, for each candidate a halting score $q_k^t = \boldsymbol{f}_Q(\boldsymbol{Z}_k^t)$ is predicted. While only the winning candidate $k_t^\star$ contributes to the task loss, the halting head is trained on all candidates. For multiple valid solutions, the halting target is defined with respect to closest label $\boldsymbol{Y}_{m_t^\star}$. The halting score itself depends only on the latent state $\boldsymbol{Z}_k^t$, whereas its training target depends on whether the decoded prediction $\mathrm{dec}(\hat{\boldsymbol{Y}}_k^t)$ matches $\boldsymbol{Y}_{m_t^\star}$. Conceptually, the halting head can be regarded as an auxiliary model for predicting solution correctness and can also be replaced by a symbolic solver at inference time.

Learned exploration. The scale of the perturbation $\boldsymbol{\epsilon}_k^t$ is learned jointly with the model, which removes the need for a search over noise levels. The learned scale grows with the number of candidates (Table 1), so the model widens its exploration when more candidates are available to absorb it. This is different to EqR (Huang et al., 2026) and PTRM (Sghaier et al., 2026), which perturb trajectories with a fixed scale and uses them for breadth scaling at inference. The effect is visible within a single trajectory: even when a single candidate is used at evaluation, xRRM revises far more digits between consecutive steps than TRM or EqR (see Fig. 2).

3 Experimental Evaluation

3.1 Experimental Setup

General Remark. Our goal is to measure the performance of xRRM under controlled conditions. Therefore we approximately match the FLOPs of EqR, TRM and xRRM during training, regardless of the number of candidates. This leads to a smaller batch size for xRRM to compensate for the additional forward passes. Similarly, the FLOPs during evaluations are also comparable for these models. Exceptions are HRM and URM. Following Wang et al. (2025) HRM is trained with fewer updates, as it both converges and overfits faster, while URM is trained for the same number of updates but needs more FLOPs because of a deeper architecture.

Data Splitting and Evaluation Setup. Each dataset contains only a small number of training samples, which are repeated over many epochs. As a result, single runs can vary considerably, especially when the test set is also randomly sampled. To reduce this variance, we draw five random pairs of training and test sets and train and evaluate every method once on each pair, using a different seed per pair. Pencil Puzzle Bench is the exception, as its test set is fixed. We report the mean and sample standard deviation over the five runs.

Evaluation Metric. Our main metric is the fully-solved rate (FSR), the fraction of problems whose complete solution is correct. The main comparison uses a single evaluation candidate, which puts the stochastic methods on an equal footing with the deterministic baselines. For the stochastic methods (EqR, PTRM and xRRM), we additionally report several inference-time test scaling results with $K_{\text{eval}} \geq 2$ candidates.

Relevant Hyperparameters. The number of training candidates K_{train} is xRRM's main hyperparameter. For each dataset, we select it using only the training data of the first run, split 70/30 into training and validation sets, so the test sets are never used for selection. The main tables report the selected K_{train}. Results for all other values are given in Section C.2. Further details on hyperparameters for our experimental evaluation are described in Section C.1.

3.2 Results and Analysis

Sudoku. Sudoku-Extreme (Wang et al., 2025) consists of difficult Sudoku puzzles with unique solutions. Following the usual protocol (Huang et al., 2026), 1000 puzzles are sampled for training

Table 2: Fully solved rate (FSR) on Sudoku-Extreme with 16 recurrent inference steps. Values are percentages shown as mean ± sample standard deviation across different seeds. Columns vary the number of evaluation candidates. The best result per number of evaluation candidates is marked bold, while the optimal number of candidates is underlined.

Model	Evaluation Candidates					
	1	2	4	8	16	32
Sudoku-Extreme						
EqR	67.8 ± 3.8	71.6 ± 3.8	75.2 ± 4.0	78.6 ± 4.5	81.9 ± 4.3	84.4 ± 4.2
HRM	59.6 ± 2.6	–	–	–	–	–
PTRM	61.7 ± 9.2	66.9 ± 6.2	70.9 ± 4.4	74.1 ± 3.7	76.5 ± 3.0	78.7 ± 3.0
TRM	61.4 ± 9.0	–	–	–	–	–
URM	80.2 ± 1.8	–	–	–	–	–
xRRM	**90.2 ± 1.8**	**95.8 ± 0.7**	**98.0 ± 0.3**	**99.0 ± 0.3**	**99.1 ± 0.7**	**99.1 ± 0.3**

and augmented 1000 times each, and 2048 puzzles are sampled for testing. Table 2 reports the results. With a single evaluation candidate, xRRM reaches a fully solved rate of 90.2 percent and thereby improves over EqR by more than 22 percentage points and over TRM by almost 29 percentage points. Increasing the number of evaluation candidates raises the fully solved rate further to 99.1 percent, which means exploration also serves as a test-time scaling axis. The gains are larger for xRRM than for EqR, which indicates that training the candidates jointly improves beyond sampling them at evaluation.

Fig. 3 illustrates the latent-state dynamics learned by the different methods on a an representative Sudoku instance. For each method, we selected two representative examples for visualization. Each plot is based on 512 latent-state trajectories, which are projected onto their first two principal components using PCA. The resulting projections provide a low-dimensional view of the inference dynamics. A possible interpretation of these trajectories is that the PTRM and EqR learn dynamics that tend to become trapped within individual attractor basins during inference. In contrast, xRRM appears to exhibit smoother transitions between attractor basins, potentially facilitating exploration of the latent state space and reducing the likelihood of getting stuck in suboptimal regions. While this interpretation is qualitative, it offers one possible explanation for the improved inference performance observed in our experiments.

Table 3: Exact accuracy on Pencil Puzzle Bench with 16 recurrent inference steps and 1 and 32 evaluation candidates. Values are percentages shown as mean ± sample standard deviation across different seeds. *All* is the pooled accuracy over all puzzles. The suffix of xRRM indicates the number of candidates used during training. The best result per puzzle and number of evaluation candidates is marked bold.

Model	Puzzle						
	Sudoku	Light Up	Nurikabe	Shakashaka	Heyawake	Tapa	All
1 Evaluation Candidate							
EqR	54.7 ± 8.7	85.0 ± 13.7	68.9 ± 9.3	**100.0 ± 0.0**	85.7 ± 10.1	26.0 ± 8.9	68.5 ± 3.7
HRM	32.0 ± 14.5	87.5 ± 0.0	75.6 ± 9.3	**100.0 ± 0.0**	82.9 ± 6.4	38.0 ± 11.0	65.9 ± 2.9
PTRM	61.3 ± 11.0	75.0 ± 12.5	57.8 ± 18.3	90.0 ± 18.1	**91.4 ± 7.8**	32.0 ± 17.9	66.9 ± 5.5
TRM	61.3 ± 11.0	75.0 ± 12.5	57.8 ± 18.3	90.0 ± 18.1	**91.4 ± 7.8**	32.0 ± 17.9	66.9 ± 5.5
URM	56.0 ± 10.1	87.5 ± 8.8	**86.7 ± 9.3**	**100.0 ± 0.0**	85.7 ± 0.0	**52.0 ± 16.4**	**76.1 ± 4.1**
xRRM	**64.0 ± 7.6**	92.5 ± 6.8	68.9 ± 12.2	**100.0 ± 0.0**	80.0 ± 7.8	40.0 ± 12.2	73.4 ± 3.6
32 Evaluation Candidates							
EqR	86.7 ± 6.7	97.5 ± 5.6	73.3 ± 12.7	**100.0 ± 0.0**	88.6 ± 6.4	38.0 ± 13.0	81.0 ± 3.4
PTRM	85.3 ± 8.7	82.5 ± 11.2	62.2 ± 16.9	91.7 ± 14.4	91.4 ± 7.8	28.0 ± 17.9	74.1 ± 6.7
xRRM	**90.7 ± 3.7**	**100.0 ± 0.0**	71.1 ± 14.9	**100.0 ± 0.0**	91.4 ± 7.8	**56.0 ± 8.9**	**85.2 ± 2.0**

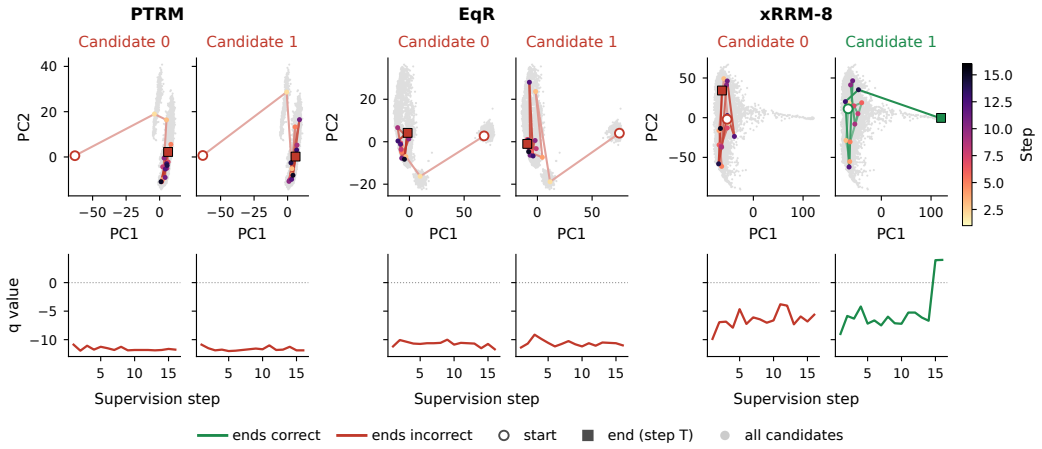


Figure 3: Latent trajectories and halting values of two candidates on Sudoku. **Top:** trajectory of the latent state z_H over the supervision steps, projected onto the first two principal components. The PCA is fit separately for each model on 512 candidates per example. The trajectories of the first two candidates are in the foreground. **Bottom:** q value per supervision step. Green and red indicate whether the candidate's prediction at the final step is correct or incorrect.

Pencil Puzzle Bench. Pencil Puzzle Bench (Waugh, 2026) consists of 62,231 logic puzzles. Each puzzle is accompanied by a verifier that checks a proposed board against the rules of the corresponding puzzle type and identifies violated constraints. We follow the experimental setting of Sghaier et al. (2026). Specifically, we use six puzzle types: Sudoku, Light Up, Nurikabe, Shakashaka, Heyawake, and Tapa. Only puzzles whose longest side is at most 10 cells are retained. The six puzzle types are combined into a single dataset, with each board padded to a 10×10 grid. For evaluation, we use the 61 puzzles of these six types contained in the official 300-puzzle benchmark set and exclude them from the training pool. We also use the same dihedral augmentations (8 reflections and rotations) as Sghaier et al. (2026). However, we do not set the augmented examples to a randomly sampled point along the solution trajectory. Firstly, because we want to allow for more exploration and not learn a fixed algorithm, and secondly, results with this augmentation are worse overall, as reported in Table C2. The results on the pencil puzzle bench depends on the specific puzzle type, as shown in Table 3. On average xRRM steadily improves over other stochastic RRMs. In the case of only one evaluation candidate, xRRM may be outperformed by URM in some of the puzzle types. However, it should be noted, that URM consumes significantly more training and inference FLOPs and might moreover profit from an inductive bias that aligns well with some of the puzzle types (i.e., locality exploited by convolutional layers).

Game-of-24 requires an arithmetic expression over four given numbers that evaluates to 24. The 1362 solvable puzzles of Lile (2024) are split into a training set of 70 percent and a test set of 30 percent. We use Game-of-24 as an example of a conditional generative problem, as the number of valid solutions per puzzle ranges from 1 to 335. Table 4 shows the results for RRMs. All baselines stay near 4 percent, whereas xRRM reaches 12.4 percent with a single evaluation candidate and 18.3 percent with 32 candidates. For a fair comparison, all methods are trained with the gradient computed for the closest of the valid solutions (Table 4, top). Other approaches, such as Yao et al. (2023), evaluate the output against the ground truth. This significantly improves the performance of xRRM (Table 4, bottom). We hypothesize that the lack of pretraining makes it hard for RRMs to evaluate whether the solution is correct. However, xRRM can still generate good candidate solutions, which we attribute to its ability to explore different valid solutions.

Alternative strategies for handling several solutions are reported in Section C.3 and perform worse throughout. The number of candidates used during training is critical on this problem: test-time scaling with additional sampling only enhances methods that already solve a substantial fraction of the puzzles. Although RRMs widely outperform much larger LLMs on Sudoku (Jolicoeur-Martineau, 2025), we find that modern LLMs with CoT perform much better on Game-of-24 ($\approx 64\%$ with gpt-oss-20b). In contrast to that GPT-4 performs quite poor, but Yao et al. (2023) managed to achieve

Table 4: Game-of-24 fully solved rate for 16 steps with both q-values (default) and a functional evaluator as selection criterion. Values are percentages shown as mean ± sample standard deviation across different seeds. Columns vary the number of evaluation candidates. The best result per number of evaluation candidates is marked bold, while the optimal number of candidates is underlined.

Model	Evaluation Candidates					
	1	2	4	8	16	32
Game-of-24 with q-values as selection criterion						
EqR	4.2 ± 1.6	4.1 ± 1.6	4.4 ± 1.7	4.2 ± 1.6	4.4 ± 1.9	4.3 ± 1.8
HRM	4.5 ± 1.7	–	–	–	–	–
PTRM	4.5 ± 0.9	4.5 ± 1.6	4.5 ± 1.1	4.6 ± 1.2	4.5 ± 1.1	4.3 ± 1.0
TRM	4.0 ± 1.3	–	–	–	–	–
URM	5.5 ± 1.6	–	–	–	–	–
xRRM	**12.4 ± 1.4**	**14.7 ± 1.7**	**16.8 ± 2.5**	**17.6 ± 2.8**	**18.0 ± 1.9**	**18.3 ± 1.3**
Game-of-24 with functional evaluator as selection criterion						
EqR	4.2 ± 1.6	4.1 ± 1.6	4.4 ± 1.7	4.2 ± 1.6	4.4 ± 1.9	4.3 ± 1.8
HRM	4.5 ± 1.7	–	–	–	–	–
PTRM	4.5 ± 0.9	4.9 ± 1.5	5.4 ± 1.3	6.3 ± 1.4	6.8 ± 1.8	7.6 ± 1.7
TRM	4.0 ± 1.3	–	–	–	–	–
URM	5.5 ± 1.6	–	–	–	–	–
xRRM	**12.4 ± 1.4**	**20.5 ± 2.2**	**33.9 ± 2.6**	**47.7 ± 4.7**	**60.4 ± 4.0**	**69.8 ± 3.7**

74% with GPT4 and their Tree of Thoughts method. However, the comparison might be inadequate, as LLMs might have been trained on test samples. Moreover, Game-of-24 is hard for models trained from scratch, since the arithmetic and the ordering of the numbers have to be learned jointly from limited examples.

4 Discussion and Limitations

Our most important observation is that training parallel stochastic trajectories improves model performance, even when evaluating with a single trajectory. On Sudoku-Extreme, xRRM-8 reaches 90.2% accuracy with a single evaluation candidate compared to 67.8% for EqR and 61.4% for TRM. Because the single-candidate setting removes any benefit from test-time ensembling, this improvement shows that the gain arises from the proposed training paradigm rather than from candidate selection. Two observations support this interpretation: First, the learned perturbation scale increases with K_{train} (Table 1), suggesting that the model can exploit higher noise levels when more candidates are available. Second, on Sudoku xRRM revises substantially more previously incorrect digits between steps than TRM or EqR (Fig. 2), indicating that parallel stochastic training leads to more effective iterative refinement.

On Game-of-24 the dominant effect is selection, not generation. The same trajectories achieve only 18.3% accuracy with the learned halting score, but 69.8% accuracy with a symbolic evaluator. Thus, the candidate pool contains a valid expression for most puzzles, and test-time scaling is effective only when the selection mechanism can reliably identify proper solutions. We attribute the weak halting head to the fact that verifying an arithmetic expression requires evaluating it, which a small model trained from scratch on less than 1000 puzzles has little chance to learn, whereas Sudoku constraint violations are locally visible.

Limitations. xRRM is sensitive to the number of training candidates K_{train}. The performance peaks at intermediate values and drops sharply for large ones (see Section C.2). Under our FLOP-matched budget, more training candidates also means fewer distinct training samples. We expect this to matter most for heterogeneous datasets such as Pencil Puzzle Bench, where six puzzle types share one batch. With small batches, each update covers only a few examples per type. This matches the steep decline on Pencil Puzzle Bench for many training candidates (from 73.4% at $K_{\text{train}} = 8$ to 34.8% at $K_{\text{train}} = 32$). It is also worth noting that the official Pencil Puzzle Bench test set (Waugh, 2026) contains only few puzzles per type, so most per-type differences lie within one standard deviation.

AI USE STATEMENT

In this work, we used generative AI tools for:

- researching relevant work
- reproducing unpublished code
- reviewing code for potential errors
- drafting figures
- polish writing, suggest better formulations (including to formulate and provide critical ingredients of mathematical claims and assist in the writing of proofs)

We have not used generative AI tools for other tasks with required disclosure. We have reviewed all AI-assisted work. Concretely, we checked proposed relevant work ourself and reviewed AI generated code, bug-fixes and text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Every number reported in the experiments is the average of five different seeds. Wherever possible, we run each of this seeds on different dataset splits. Moreover, we provide a detailed description of the training and evaluation procedure as well as the model in Section B to facilitate reproducibility.

REFERENCES

L B Almeida. A learning rule for asynchronous perceptrons with feed-back in a combinatorial environment. In *IEEE First International Conference on Neural Networks*, volume 2, pp. 609–618. IEEE, 1987.

Junyeob Baek, Mingyu Jo, Minsu Kim, Mengye Ren, Yoshua Bengio, and Sungjin Ahn. Generative recursive reasoning. *arXiv preprint arXiv:2605.19376*, 2026.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2019.

Maxence Ernoult, Julie Grollier, Damien Querlioz, Yoshua Bengio, and Benjamin Scellier. Updates of equilibrium prop match gradients of backprop through time in an rnn with static input. *Advances in Neural Information Processing Systems*, 32, 2019.

Richard Freinschlag, Timo Bertram, Erich Kobler, Andreas Mayr, and Günter Klambauer. Symbol-equivariant recurrent reasoning models. *arXiv preprint arXiv:2603.02193*, 2026.

Zitian Gao, Lynx Chen, Yihao Xiao, He Xing, Ran Tao, Haoming Luo, Joey Zhou, and Bryan Dai. Universal reasoning model. *arXiv preprint arXiv:2512.14693*, 2025.

Alexi Gladstone, Heng Ji, and Yilun Du. Explorative modeling: Unlocking a third pretraining axis and end-to-end generation. *arXiv preprint arXiv:2607.27372*, 2026.

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

Abner Guzman-Rivera, Dhruv Batra, and Pushmeet Kohli. Multiple choice learning: Learning to produce multiple structured outputs. *Advances in neural information processing systems*, 25, 2012.

Abner Guzman-Rivera, Pushmeet Kohli, Dhruv Batra, and Rob Rutenbar. Efficiently enforcing diversity in multi-output structured prediction. In *Artificial Intelligence and Statistics*, pp. 284–292. PMLR, 2014.

Rishi Hazra, Gabriele Venturato, Pedro Zuidberg Dos Martires, and Luc De Raedt. Can large language models reason? a characterization via 3-SAT. *arXiv preprint arXiv:2408.07215*, 2024.

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

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

Stefan Lee, Senthil Purushwalkam Shiva Prakash, Michael Cogswell, Viresh Ranjan, David Crandall, and Dhruv Batra. Stochastic multiple choice learning for training diverse deep ensembles. *Advances in Neural Information Processing Systems*, 29, 2016.

Nathan Lile. Math twenty four (24s game) dataset. https://huggingface.co/datasets/nlile/24-game, 2024. Hugging Face Dataset repository.

Osama Makansi, Eddy Ilg, Ozgun Cicek, and Thomas Brox. Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7144–7153, 2019.

Alan Malek, Jiawei Ge, Nevena Lazic, Chi Jin, András György, and Csaba Szepesvári. Frontier llms still struggle with simple reasoning tasks. *arXiv preprint arXiv:2507.07313*, 2025.

David Perera, Victor Letzelter, Théo Mariotte, Adrien Cortés, Mickael Chen, Slim Essid, and Gaël Richard. Annealed multiple choice learning: Overcoming limitations of winner-takes-all with annealing. *Advances in neural information processing systems*, 37:10541–10574, 2024.

Fernando Pineda. Generalization of back propagation to recurrent and higher order neural networks. In *Neural information processing systems*, 1987.

Christian Rupprecht, Iro Laina, Robert DiPietro, Maximilian Baust, Federico Tombari, Nassir Navab, and Gregory D Hager. Learning in an uncertain world: Representing ambiguity through multiple hypotheses. In *2017 IEEE international conference on computer vision (ICCV)*, pp. 3611–3620. IEEE, 2017.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Jeffrey Seely, Yuki Imajuku, Tianyu Zhao, Edoardo Cetin, and Llion Jones. Sudoku-Bench: Evaluating creative reasoning with Sudoku variants. *arXiv preprint arXiv:2505.16135*, 2025.

Amin Sghaier, Ali Parviz, and Alexia Jolicoeur-Martineau. Probabilistic tiny recursive model. *arXiv preprint arXiv:2605.19943*, 2026.

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.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

APPENDIX

A Notation **14**

B xRRM: Algorithm **15**

C Experimental Details and Additional Results **18**

C.1 Hyperparameters 18

C.2 Variation in the Number of Training Candidates 18

C.3 Alternative Optimization Targets for Conditional Generative Problems 21

A NOTATION

Definition	Symbol	Type
Scalars and indices		
feature dimension	D	$\mathbb{N}$
number of positions (grid points)	I	$\mathbb{N}$
number of deep supervision steps	T	$\mathbb{N}$
deep supervision step index	t	$\{1, \dots, T\}$
number of candidate trajectories	K	$\mathbb{N}$
candidate index	k	$\{1, \dots, K\}$
candidates during training / evaluation	$K_{\text{train}}, K_{\text{eval}}$	$\mathbb{N}$
winning candidate at step t	$k_t^\star$	$\{1, \dots, K\}$
number of valid target solutions	M	$\mathbb{N}$
target index	m	$\{1, \dots, M\}$
winning target at step t	$m_t^\star$	$\{1, \dots, M\}$
recurrent cycles per supervision step	R	$\mathbb{N}$
low-level updates per cycle	n	$\mathbb{N}$
halting score of candidate k at step t	q_k^t	$\mathbb{R}$
Data and predictions		
input problem instance	$\boldsymbol{X}$	$\mathcal{X}$
target solution	$\boldsymbol{Y}$	$\mathcal{Y}$
m-th valid target solution	$\boldsymbol{Y}_m$	$\mathcal{Y}$
logits of candidate k at step t	$\hat{\boldsymbol{Y}}_k^t$	$\mathbb{R}^{I \times V}$
decoded solution of candidate k at step t	$\mathrm{dec}(\hat{\boldsymbol{Y}}_k^t)$	$\mathcal{Y}$
task embedding of the input	$\boldsymbol{E} = E(\boldsymbol{X})$	$\mathbb{R}^{I \times D}$
high-level recurrent state	$\boldsymbol{Z}_{H}{}_k^t$	$\mathbb{R}^{I \times D}$
low-level recurrent state	$\boldsymbol{Z}_{L}{}_k^t$	$\mathbb{R}^{I \times D}$
random perturbation of candidate k at step t	$\boldsymbol{\epsilon}_k^t$	$\mathbb{R}^{I \times D}$
Maps and models		
input embedding map	E	$\mathcal{X} \to \mathbb{R}^{I \times D}$
recurrent block (one supervision step)	$\mathcal{G}$	$(\boldsymbol{E}, \boldsymbol{Z}, \boldsymbol{\epsilon}) \mapsto \boldsymbol{Z}$
shared recurrent update block	$\boldsymbol{f}_\theta$	$\mathbb{R}^{I \times D} \to \mathbb{R}^{I \times D}$
output map, state to logits	$\boldsymbol{f}_O$	$\mathbb{R}^{I \times D} \to \mathbb{R}^{I \times V}$
halting head	$\boldsymbol{f}_Q$	$\mathbb{R}^{I \times D} \to \mathbb{R}$
model parameters	$\boldsymbol{\theta}$	$\mathcal{W}$
decoding, logits to discrete solution	$\mathrm{dec}(\cdot)$	$\mathbb{R}^{I \times V} \to \mathcal{Y}$
normalization inside the perturbation	$\mathrm{Norm}(\cdot)$	$\mathbb{R}^{I \times D} \to \mathbb{R}^{I \times D}$
stop-gradient operator	$\mathrm{sg}(\cdot)$	identity, zero Jacobian
Losses		
task loss between prediction and target	$L(\hat{\boldsymbol{Y}}, \boldsymbol{Y})$	$\mathbb{R}_{\geq 0}$
halting loss	$L_Q(q, \cdot)$	$\mathbb{R}_{\geq 0}$
training objective at step t	$\mathcal{L}^t$	$\mathbb{R}_{\geq 0}$

B xRRM: ALGORITHM

Algorithm 1 One deep supervision step $\mathcal{G}$ of xRRM. At the beginning of each internal cycle, the high- and low-level latent states are perturbed by candidate-specific noise. The perturbation magnitude is controlled by the learned weight matrices $\boldsymbol{W}_H$ and $\boldsymbol{W}_L$. Only the final internal cycle carries gradients. $\mathrm{sg}(\cdot)$ denotes the stop-gradient operator.

function PERTURB($\boldsymbol{Z}_H, \boldsymbol{Z}_L, \boldsymbol{\epsilon}_H, \boldsymbol{\epsilon}_L$)
  $\boldsymbol{Z}_H \leftarrow \mathrm{Norm}(\boldsymbol{Z}_H + \boldsymbol{\epsilon}_H \odot \boldsymbol{Z}_H \boldsymbol{W}_H)$
  $\boldsymbol{Z}_L \leftarrow \mathrm{Norm}(\boldsymbol{Z}_L + \boldsymbol{\epsilon}_L \odot \boldsymbol{Z}_L \boldsymbol{W}_L)$
  return $\boldsymbol{Z}_H, \boldsymbol{Z}_L$
end function

function LATENTRECURSION($\boldsymbol{E}, \boldsymbol{Z}_H, \boldsymbol{Z}_L$)
  for $j = 1$ **to** n **do**
    $\boldsymbol{Z}_L \leftarrow \mathcal{G}_\theta(\boldsymbol{E} + \boldsymbol{Z}_H + \boldsymbol{Z}_L)$
  end for
  $\boldsymbol{Z}_H \leftarrow \mathcal{G}_\theta(\boldsymbol{Z}_H + \boldsymbol{Z}_L)$
  return $\boldsymbol{Z}_H, \boldsymbol{Z}_L$
end function

function REASONINGSTEP($\boldsymbol{E}, \boldsymbol{Z}_H, \boldsymbol{Z}_L, \boldsymbol{\epsilon}$)
  for $r = 1$ **to** $R - 1$ **do** ▷ without gradients
    $\boldsymbol{Z}_H, \boldsymbol{Z}_L \leftarrow$ PERTURB($\boldsymbol{Z}_H, \boldsymbol{Z}_L, \boldsymbol{\epsilon}_{H,r}, \boldsymbol{\epsilon}_{L,r}$)
    $\boldsymbol{Z}_H, \boldsymbol{Z}_L \leftarrow$ LATENTRECURSION($\boldsymbol{E}, \boldsymbol{Z}_H, \boldsymbol{Z}_L$)
  end for
  $\boldsymbol{Z}_H, \boldsymbol{Z}_L \leftarrow$ PERTURB($\boldsymbol{Z}_H, \boldsymbol{Z}_L, \boldsymbol{\epsilon}_{H,R}, \boldsymbol{\epsilon}_{L,R}$) ▷ with gradients
  $\boldsymbol{Z}_H, \boldsymbol{Z}_L \leftarrow$ LATENTRECURSION($\boldsymbol{E}, \boldsymbol{Z}_H, \boldsymbol{Z}_L$)
  $\hat{\boldsymbol{Y}} \leftarrow \boldsymbol{f}_O(\boldsymbol{Z}_H)$
  $q \leftarrow \boldsymbol{f}_Q(\boldsymbol{Z}_H)$
  return $\big(\mathrm{sg}(\boldsymbol{Z}_H), \mathrm{sg}(\boldsymbol{Z}_L)\big), \hat{\boldsymbol{Y}}, q$
end function

Algorithm 2 Training an Explorative Recurrent Reasoning Model (xRRM). At every deep supervision step, K_{train} candidate trajectories are advanced without gradients using independent random perturbations. Each candidate is paired with the target solution it matches best; the candidate with the lowest task loss is then replayed with gradients using the same perturbation. The halting head is trained on all candidates, each against its own target. The carry of the exploration pass is kept, so all candidate trajectories are retained for the subsequent deep supervision step.

function SELECTWINNER($\{\hat{\boldsymbol{Y}}_k\}_{k=1}^{K_{\text{train}}}, \{\boldsymbol{Y}_m\}_{m=1}^{M}$)
  for $k = 1$ **to** K_{train} **do**
    $m_k^\star \leftarrow \arg\min_{m\in\{1,\dots,M\}} L\Big(\hat{\boldsymbol{Y}}_k, \boldsymbol{Y}_m\Big)$ ▷ evaluates the loss for all M targets
    $L_k \leftarrow L\Big(\hat{\boldsymbol{Y}}_k, \boldsymbol{Y}_{m_k^\star}\Big)$
  end for
  $k^\star \leftarrow \arg\min_{k\in\{1,\dots,K_{\text{train}}\}} L_k$
  return $k^\star, \{m_k^\star\}_{k=1}^{K_{\text{train}}}$
end function

function EFFICIENTSELECTWINNER($\{\hat{\boldsymbol{Y}}_k\}_{k=1}^{K_{\text{train}}}, \{\boldsymbol{Y}_m\}_{m=1}^{M}$)
  for $k = 1$ **to** K_{train} **do**
    $m_k^\star \leftarrow \arg\max_{m\in\{1,\dots,M\}} \sum_{i=1}^{I} \mathbf{1}\Big[\text{dec}(\hat{\boldsymbol{Y}}_k)_i = (\boldsymbol{Y}_m)_i\Big]$ ▷ target with most agreeing positions
    $L_k \leftarrow L\Big(\hat{\boldsymbol{Y}}_k, \boldsymbol{Y}_{m_k^\star}\Big)$ ▷ loss evaluated once per candidate
  end for
  $k^\star \leftarrow \arg\min_{k\in\{1,\dots,K_{\text{train}}\}} L_k$
  return $k^\star, \{m_k^\star\}_{k=1}^{K_{\text{train}}}$
end function

for $(\boldsymbol{X}, \{\boldsymbol{Y}_m\}_{m=1}^{M})$ **in** training set **do**
  $\boldsymbol{E} \leftarrow E(\boldsymbol{X})$
  Initialize $\boldsymbol{Z}_{H_k}^{0}, \boldsymbol{Z}_{L_k}^{0} \sim \mathcal{N}(0,1)$ for $k = 1, \dots, K_{\text{train}}$
  for $t = 1$ **to** T **do**
    for $k = 1$ **to** K_{train} **do**
      Sample $\boldsymbol{\epsilon}_k^t \equiv \{\boldsymbol{\epsilon}_{H,k,r}^t, \boldsymbol{\epsilon}_{L,k,r}^t\}_{r=1}^{R}$ entrywise $\sim \mathcal{N}(0,1)$ ▷ one draw per internal cycle
      $(\boldsymbol{Z}_H{}_k^t, \boldsymbol{Z}_L{}_k^t), \hat{\boldsymbol{Y}}_k^t, \cdot \leftarrow \mathcal{G}\left(\boldsymbol{E}, \boldsymbol{Z}_H{}_k^{t-1}, \boldsymbol{Z}_L{}_k^{t-1}; \boldsymbol{\epsilon}_k^t\right)$ ▷ without gradients
    end for
    $\Big(k_t^\star, \{m_{t,k}^\star\}_{k=1}^{K_{\text{train}}}\Big) \leftarrow$ (EFFICIENT)SELECTWINNER($\{\hat{\boldsymbol{Y}}_k^t\}_{k=1}^{K_{\text{train}}}, \{\boldsymbol{Y}_m\}_{m=1}^{M}$)
    $m_t^\star \leftarrow m_{t,k_t^\star}^\star$
    for $k = 1$ **to** K_{train} **do**
      $q_k^t \leftarrow f_Q(\text{sg}(\boldsymbol{Z}_H{}_k^t))$ ▷ gradients through f_Q only
    end for
    $\cdot, \hat{\boldsymbol{Y}}^t, \cdot \leftarrow \mathcal{G}\left(\boldsymbol{E}, \boldsymbol{Z}_H{}_{k_t^\star}^{t-1}, \boldsymbol{Z}_L{}_{k_t^\star}^{t-1}; \boldsymbol{\epsilon}_{k_t^\star}^t\right)$ ▷ replay winner with gradients
    $\mathcal{L}_{\text{task}}^t \leftarrow L\Big(\hat{\boldsymbol{Y}}^t, \boldsymbol{Y}_{m_t^\star}\Big)$
    $\mathcal{L}_Q^t \leftarrow \frac{1}{K_{\text{train}}} \sum_{k=1}^{K_{\text{train}}} L_Q\Big(q_k^t, \mathbf{1}\Big[\text{dec}(\hat{\boldsymbol{Y}}_k^t) = \boldsymbol{Y}_{m_{t,k}^\star}\Big]\Big)$ ▷ each candidate against its own target
    $\mathcal{L}^t \leftarrow \mathcal{L}_{\text{task}}^t + \mathcal{L}_Q^t$
    Update $\boldsymbol{\theta}$ using $\nabla_{\boldsymbol{\theta}} \mathcal{L}^t$
    if $q_{k_t^\star}^t > 0$ **then break**
  end for
end for

Algorithm 3 Inference with K_{eval} candidate trajectories. Each candidate follows an independently perturbed recurrent trajectory. Since the target solution is unavailable, the task loss cannot be used for candidate selection. The final prediction is therefore selected using the learned correctness/halting score q.

for $\boldsymbol{X}$ **in** test set **do**
  $\boldsymbol{E} \leftarrow E(\boldsymbol{X})$
  Initialize $\boldsymbol{Z}_{H_k}^{\,0}, \boldsymbol{Z}_{L_k}^{\,0} \sim \mathcal{N}(0,1)$ for $k = 1, \ldots, K_{\text{eval}}$
  for $t = 1$ **to** T **do**
    for $k = 1$ **to** K_{eval} **do**
      Sample $\boldsymbol{\epsilon}_k^t \sim \mathcal{N}(0,1)$
      $(\boldsymbol{Z}_{H_k}^{\,t}, \boldsymbol{Z}_{L_k}^{\,t}), \hat{\boldsymbol{Y}}_k^t, q_k^t \leftarrow \mathcal{G}\left(\boldsymbol{E}, \boldsymbol{Z}_{H_k}^{\,t-1}, \boldsymbol{Z}_{L_k}^{\,t-1}; \boldsymbol{\epsilon}_k^t\right)$
    end for
  end for
  $k^\star \leftarrow \arg\max_{k \in \{1,\ldots,K_{\text{eval}}\}} q_k^T$
  return $\operatorname{dec}\left(\hat{\boldsymbol{Y}}_{k^\star}^T\right)$
end for

C Experimental Details and Additional Results

C.1 Hyperparameters

xRRM has in general $3(K_{train}+1)$ forward passes per backward pass. We assume that every backward pass costs approximately two times more than a forward pass and adjust the batch size accordingly, to approximately match training FLOPs. Table C1 shows the actual measured FLOPs and memory required for one deep supervision step. As the backward pass dominates the memory consumption, xRRMs with many candidates are highly memory efficient. For our experiments we take the original hyperparameters of TRM (Jolicoeur-Martineau, 2025) for xRRM and adjust the batch size. As the number of epochs also depends on the batch size in the original code base, we adjust the hyperparameter epochs as well, so that every model trains for the same number of steps.

We only use the attention variant of TRM (Jolicoeur-Martineau, 2025). The Sudoku-Extreme results are worse than in the original paper. We attribute this to the high variance between single runs.

HRM (Wang et al., 2025) and is retrained with the original hyperparameters with the exception that an Exponential Moving Average weight update (EMA) is applied. This might explain the improved performance compared to the original paper.

For URM (Gao et al., 2025) we apply the same number of updates as for the other models (EqR, TRM, xRRM) and do not change any other hyperparameters compared to the original implementation.

The architecture and training procedure of PTRM (Sghaier et al., 2026) is exactly the same as of TRM (Jolicoeur-Martineau, 2025). Therefore, we use the trained TRM models and set the noise level σ to 0.2 during evaluation. The first evaluation candidate of PTRM is, as in the original paper (Sghaier et al., 2026) without noise and therefore a common TRM. For both models, TRM and PTRM, we only use the attention variant. While the MLP-mixer variant achieves better results for Sudoku, it delivers inferior results on other datasets.

Table C1: Training-time recurrent computation per deep supervision step. TFLOPs and Memory consumption are measured on one batch of Sudoku-Extreme.

Model	Forward `H_cycles`	Backward `H_cycles`	batch size	TFLOPs	Memory (GB)
TRM	3	1	768	36.617	21.2
xRRM-1	6	1	480	36.962	13.6
xRRM-2	9	1	349	36.966	10.2
xRRM-4	15	1	226	37.008	6.8
xRRM-8	27	1	132	36.883	4.2
xRRM-16	51	1	72	36.774	2.6
xRRM-32	99	1	38	36.990	2.2

C.2 Variation in the Number of Training Candidates

xRRM is sensitive to the number of training candidates, K_{train}. This dependence may partly stem from our fixed training-FLOP budget: although the FLOPs of individual runs were matched, configurations with more candidates effectively process fewer distinct training samples. Consequently, the additional hyperparameter K_{train} may itself require additional compute to tune reliably. In contrast to EqR and PTRM, however, xRRM learns the scale of the injected noise, thereby eliminating an additional noise-scale hyperparameter.

Table C2: Complete results with hyperparameter ablation for xRRM. The suffix of xRRM indicates the number of candidates used during training. For Pencil Puzzle bench the results with trajectory augmentation as in Sghaier et al. (2026) are included at the bottom.

Model	Evaluation Candidates					
	1	2	4	8	16	32
Sudoku-Extreme						
EqR	67.8 ± 3.8	71.6 ± 3.8	75.2 ± 4.0	78.6 ± 4.5	81.9 ± 4.3	84.4 ± 4.2
HRM	59.6 ± 2.6	–	–	–	–	–
PTRM	61.7 ± 9.2	66.9 ± 6.2	70.9 ± 4.4	74.1 ± 3.7	76.5 ± 3.0	78.7 ± 3.0
TRM	61.4 ± 9.0	–	–	–	–	–
URM	80.2 ± 1.8	–	–	–	–	–
xRRM-1	48.6 ± 6.6	53.3 ± 6.9	57.1 ± 6.9	61.0 ± 6.4	63.6 ± 5.9	66.7 ± 5.2
xRRM-2	76.4 ± 2.3	81.2 ± 1.5	85.0 ± 1.3	88.7 ± 1.2	90.5 ± 1.6	92.2 ± 1.6
xRRM-4	87.5 ± 9.2	91.5 ± 9.0	93.4 ± 9.0	93.7 ± 9.7	94.4 ± 8.8	95.2 ± 8.2
xRRM-8	**90.2 ± 1.8**	**95.8 ± 0.7**	**98.0 ± 0.3**	**99.0 ± 0.3**	**99.1 ± 0.7**	**99.1 ± 0.3**
xRRM-16	74.7 ± 1.7	88.8 ± 1.0	95.8 ± 0.7	98.4 ± 0.8	99.0 ± 0.8	98.9 ± 1.2
xRRM-32	54.8 ± 4.7	69.8 ± 5.4	83.1 ± 4.7	90.8 ± 4.0	94.6 ± 2.4	96.3 ± 1.6
Pencil Puzzle Bench						
EqR	68.5 ± 3.7	72.8 ± 3.0	74.4 ± 3.2	78.0 ± 5.0	79.7 ± 3.4	81.0 ± 3.4
HRM	65.9 ± 2.9	–	–	–	–	–
PTRM	66.9 ± 5.5	69.5 ± 5.3	70.5 ± 4.6	72.5 ± 5.1	73.1 ± 4.6	74.1 ± 6.7
TRM	66.9 ± 5.5	–	–	–	–	–
URM	**76.1 ± 4.1**	–	–	–	–	–
xRRM-1	58.0 ± 1.9	60.0 ± 2.2	63.3 ± 1.5	64.9 ± 2.2	64.6 ± 2.7	64.9 ± 2.5
xRRM-2	63.0 ± 3.8	66.6 ± 3.6	68.5 ± 2.7	69.5 ± 4.6	70.5 ± 3.5	71.8 ± 5.2
xRRM-4	70.5 ± 3.5	72.8 ± 3.8	75.7 ± 3.2	79.3 ± 3.6	78.7 ± 4.2	79.0 ± 4.1
xRRM-8	73.4 ± 3.6	**78.0 ± 3.8**	**79.7 ± 4.6**	**82.3 ± 3.7**	**84.3 ± 4.3**	**85.2 ± 2.0**
xRRM-16	56.4 ± 3.2	63.6 ± 1.8	68.2 ± 3.8	67.5 ± 3.7	73.4 ± 2.7	71.8 ± 3.2
xRRM-32	34.8 ± 3.9	38.7 ± 3.0	42.3 ± 2.1	45.6 ± 5.4	46.6 ± 3.4	46.9 ± 3.4
Pencil Puzzle Bench with trajectory augmentation						
EqR	64.9 ± 3.4	67.9 ± 1.5	71.5 ± 3.0	73.8 ± 3.8	75.4 ± 3.1	75.7 ± 3.9
HRM	59.7 ± 1.9	–	–	–	–	–
PTRM	62.0 ± 4.2	62.0 ± 6.0	64.3 ± 4.4	64.3 ± 4.1	66.2 ± 4.9	67.5 ± 2.7
TRM	62.0 ± 4.2	–	–	–	–	–
URM	**69.8 ± 1.5**	–	–	–	–	–
xRRM-1	59.7 ± 3.8	60.7 ± 2.6	62.6 ± 3.6	65.9 ± 4.7	66.6 ± 5.0	66.6 ± 8.4
xRRM-2	64.6 ± 4.1	66.6 ± 2.5	69.2 ± 2.7	69.2 ± 3.6	69.2 ± 1.8	72.1 ± 3.5
xRRM-4	67.5 ± 8.1	66.9 ± 7.1	**73.8 ± 6.5**	**75.7 ± 6.9**	**76.1 ± 6.0**	76.4 ± 5.0
xRRM-8	64.2 ± 4.7	**68.9 ± 3.4**	71.6 ± 4.0	73.5 ± 3.8	75.1 ± 5.9	**77.0 ± 6.0**
xRRM-16	53.1 ± 6.3	55.4 ± 4.2	56.7 ± 3.0	63.3 ± 4.1	60.3 ± 3.6	64.6 ± 2.2
xRRM-32	26.6 ± 2.4	31.1 ± 2.8	29.5 ± 5.6	30.8 ± 2.7	36.1 ± 5.1	36.4 ± 4.5
Game-of-24						
EqR	4.2 ± 1.6	4.1 ± 1.6	4.4 ± 1.7	4.2 ± 1.6	4.4 ± 1.9	4.3 ± 1.8
HRM	4.5 ± 1.7	–	–	–	–	–
PTRM	4.5 ± 0.9	4.5 ± 1.6	4.5 ± 1.1	4.6 ± 1.2	4.5 ± 1.1	4.3 ± 1.0
TRM	4.0 ± 1.3	–	–	–	–	–
URM	5.5 ± 1.6	–	–	–	–	–
xRRM-1	3.4 ± 0.3	3.6 ± 0.7	3.8 ± 0.4	3.8 ± 0.4	4.0 ± 0.4	3.9 ± 0.2
xRRM-2	3.1 ± 1.2	3.4 ± 1.0	3.5 ± 0.5	3.5 ± 1.1	3.8 ± 0.4	3.8 ± 0.8
xRRM-4	3.8 ± 1.3	3.0 ± 0.8	4.1 ± 0.4	3.8 ± 1.3	4.1 ± 1.3	3.5 ± 1.0
xRRM-8	8.4 ± 3.4	9.3 ± 2.5	9.3 ± 2.6	9.8 ± 2.8	9.9 ± 2.7	10.0 ± 1.3
xRRM-16	**12.4 ± 1.4**	**14.7 ± 1.7**	**16.8 ± 2.5**	**17.6 ± 2.8**	**18.0 ± 1.9**	**18.3 ± 1.3**
xRRM-32	9.0 ± 1.2	9.1 ± 0.3	9.4 ± 1.2	12.0 ± 1.4	12.0 ± 1.6	11.5 ± 1.4

Table C3: Exact accuracy on Pencil Puzzle Bench with 16 recurrent inference steps and 1 and 32 evaluation candidates. Values are percentages shown as mean ± sample standard deviation across different seeds. *All* is the pooled accuracy over all puzzles. The suffix of xRRM indicates the number of candidates used during training. The best result per puzzle and number of evaluation candidates is marked bold.

Model	Puzzle						
	Sudoku	Light Up	Nurikabe	Shakashaka	Heyawake	Tapa	All
1 Evaluation Candidate							
EqR	54.7 ± 8.7	85.0 ± 13.7	68.9 ± 9.3	**100.0 ± 0.0**	85.7 ± 10.1	26.0 ± 8.9	68.5 ± 3.7
HRM	32.0 ± 14.5	87.5 ± 0.0	75.6 ± 9.3	**100.0 ± 0.0**	82.9 ± 6.4	38.0 ± 11.0	65.9 ± 2.9
PTRM	61.3 ± 11.0	75.0 ± 12.5	57.8 ± 18.3	90.0 ± 18.1	**91.4 ± 7.8**	32.0 ± 17.9	66.9 ± 5.5
TRM	61.3 ± 11.0	75.0 ± 12.5	57.8 ± 18.3	90.0 ± 18.1	**91.4 ± 7.8**	32.0 ± 17.9	66.9 ± 5.5
URM	56.0 ± 10.1	87.5 ± 8.8	**86.7 ± 9.3**	**100.0 ± 0.0**	85.7 ± 0.0	**52.0 ± 16.4**	**76.1 ± 4.1**
xRRM-1	13.3 ± 10.5	85.0 ± 10.5	66.7 ± 7.9	98.3 ± 3.7	77.1 ± 7.8	34.0 ± 11.4	58.0 ± 1.9
xRRM-2	24.0 ± 13.8	95.0 ± 6.8	66.7 ± 13.6	**100.0 ± 0.0**	77.1 ± 7.8	38.0 ± 11.0	63.0 ± 3.8
xRRM-4	52.0 ± 12.8	**97.5 ± 5.6**	64.4 ± 14.5	93.3 ± 7.0	88.6 ± 6.4	42.0 ± 11.0	70.5 ± 3.5
xRRM-8	**64.0 ± 7.6**	92.5 ± 6.8	68.9 ± 12.2	**100.0 ± 0.0**	80.0 ± 7.8	40.0 ± 12.2	73.4 ± 3.6
xRRM-16	33.3 ± 9.4	75.0 ± 19.8	46.7 ± 14.5	96.7 ± 4.6	68.6 ± 6.4	28.0 ± 4.5	56.4 ± 3.2
xRRM-32	2.7 ± 3.7	55.0 ± 14.3	22.2 ± 7.9	91.7 ± 5.9	42.9 ± 24.7	4.0 ± 5.5	34.8 ± 3.9
32 Evaluation Candidates							
EqR	86.7 ± 6.7	97.5 ± 5.6	73.3 ± 12.7	**100.0 ± 0.0**	88.6 ± 6.4	38.0 ± 13.0	81.0 ± 3.4
PTRM	85.3 ± 8.7	82.5 ± 11.2	62.2 ± 16.9	91.7 ± 14.4	91.4 ± 7.8	28.0 ± 17.9	74.1 ± 6.7
xRRM-1	32.0 ± 14.5	90.0 ± 5.6	68.9 ± 9.3	98.3 ± 3.7	82.9 ± 12.0	38.0 ± 8.4	64.9 ± 2.5
xRRM-2	52.0 ± 14.5	95.0 ± 6.8	71.1 ± 16.9	**100.0 ± 0.0**	88.6 ± 6.4	38.0 ± 4.5	71.8 ± 5.2
xRRM-4	69.3 ± 12.1	**100.0 ± 0.0**	**80.0 ± 9.3**	95.0 ± 7.5	**94.3 ± 7.8**	46.0 ± 5.5	79.0 ± 4.1
xRRM-8	**90.7 ± 3.7**	**100.0 ± 0.0**	71.1 ± 14.9	**100.0 ± 0.0**	91.4 ± 7.8	**56.0 ± 8.9**	**85.2 ± 2.0**
xRRM-16	64.0 ± 13.0	97.5 ± 5.6	55.6 ± 7.9	98.3 ± 3.7	77.1 ± 16.3	42.0 ± 17.9	71.8 ± 3.2
xRRM-32	21.3 ± 9.9	75.0 ± 8.8	35.6 ± 9.3	98.3 ± 3.7	51.4 ± 7.8	8.0 ± 8.4	46.9 ± 3.4

C.3 Alternative Optimization Targets for Conditional Generative Problems

Besides Winner-Takes-All approaches there are two straightforward approaches:

- select one random solution as label
- duplicate the input and add all solutions as labels to the dataset

Both are unsuitable for RRMs. While the first approach filters a lot of information, the second approach is unsuitable because it interferes with the stopping criterion, as the model can never be certain about its output. Nevertheless, we tried both variants and report the results below.

Table C4: Ablation for Game-of-24 with one randomly selected label for 16 steps. Values are percentages shown as mean ± sample standard deviation across different seeds. Columns vary the number of evaluation candidates. The suffix of xRRM indicates the number of candidates used during training. The best result for each number of evaluation candidates is marked bold, while the optimal number of candidates is underlined.

Model	Evaluation Candidates					
	1	2	4	8	16	32
EqR	1.9 ± 0.6	1.9 ± 0.4	2.0 ± 0.3	2.0 ± 0.3	1.8 ± 0.4	1.9 ± 0.5
HRM	2.8 ± 0.6	–	–	–	–	–
TRM	1.8 ± 0.4	–	–	–	–	–
xRRM-1	1.3 ± 0.2	1.1 ± 0.2	1.4 ± 0.4	1.6 ± 0.3	1.3 ± 0.3	1.7 ± 0.2
xRRM-2	2.3 ± 0.7	2.2 ± 0.8	2.5 ± 0.8	2.7 ± 0.8	2.7 ± 0.5	2.7 ± 0.5
xRRM-4	2.6 ± 0.7	2.5 ± 0.9	**3.5 ± 0.7**	3.7 ± 0.8	4.1 ± 0.8	4.0 ± 0.8
xRRM-8	**2.9 ± 0.4**	**2.8 ± 1.3**	3.4 ± 0.5	**4.3 ± 1.0**	**4.5 ± 0.8**	**4.3 ± 1.2**
xRRM-16	1.7 ± 0.8	2.6 ± 0.8	2.2 ± 1.1	3.1 ± 1.3	2.8 ± 1.5	3.0 ± 1.2
xRRM-32	1.3 ± 0.6	1.5 ± 0.7	1.8 ± 0.6	2.2 ± 0.8	1.9 ± 0.6	2.0 ± 0.3

Table C5: Ablation for Game-of-24 with duplicated inputs for 16 steps. Values are percentages shown as mean ± sample standard deviation across different seeds. Columns vary the number of evaluation candidates. The suffix of xRRM indicates the number of candidates used during training. The best result for each number of evaluation candidates is marked bold, while the optimal number of candidates is underlined.

Model	Evaluation Candidates					
	1	2	4	8	16	32
EqR	0.6 ± 0.5	0.6 ± 0.5	0.5 ± 0.4	0.6 ± 0.5	0.6 ± 0.5	0.7 ± 0.5
HRM	0.7 ± 0.7	–	–	–	–	–
TRM	1.1 ± 0.4	–	–	–	–	–
xRRM-1	0.4 ± 0.3	0.5 ± 0.3	0.5 ± 0.3	0.4 ± 0.3	0.4 ± 0.3	0.4 ± 0.3
xRRM-2	1.3 ± 0.5	1.6 ± 0.8	2.1 ± 1.2	2.0 ± 1.0	2.2 ± 1.2	2.2 ± 1.3
xRRM-4	**2.1 ± 0.4**	**2.3 ± 1.0**	**2.8 ± 0.7**	**3.0 ± 0.7**	3.1 ± 0.7	3.2 ± 1.2
xRRM-8	2.0 ± 1.0	2.1 ± 0.6	2.7 ± 0.7	2.6 ± 0.2	**3.3 ± 1.1**	**3.3 ± 1.3**
xRRM-16	1.8 ± 1.4	1.9 ± 0.9	2.1 ± 1.0	2.3 ± 0.9	2.4 ± 0.9	2.1 ± 0.7
xRRM-32	0.9 ± 0.6	1.1 ± 0.5	1.0 ± 0.4	1.4 ± 0.5	1.6 ± 0.7	1.0 ± 0.6