Appendix A Hardware

Experiments on Sudoku Extreme were conducted using two A100 GPUs (80GB). The experiments on Maze-Hard for the RIMA model were also performed on two A100 (80GB) with batch size of 128. All other experiments used either four A100 GPUs (80GB) or four H100 GPUs. For ARC-AGI-1 and Sudoku Extreme, we reproduced the original TRM results using the authors' codebase and reported a batch size of 768. For ARC-AGI-2 and Maze-Hard, we report the results from Jolicoeur-Martineau [1] due to computational constraints. The training of the HybRIM was performed on a single A100 GPU (80GB).

Appendix B Implementation details

Neural reasoning. We keep the training setup identical to Jolicoeur-Martineau [1]. Specifically, we use a learning rate of 10^{-4} with 2,000 warm-up steps. The learning rate for input embeddings is 10^{-2} with weight decay 0.1. The batch size is 768 and the embedding dimension is 512. The optimizer is Adam-atan2.

Tabular reasoning. For TabRIM we used $\epsilon = 0.05$, 5 burn-in samples, and 10 post-burn-in samples. For TabHybRIM we follow the training procedure of Koch et al. [2].

Parameter counts. Table C1 reports exact parameter counts per model and benchmark. RIMA adds approximately 525K parameters over the TRM baseline ($\sim$10% increase), while RIMFormer is larger due to its Transformer-based **Reweighter**. The performance gains of RIMA over SimRIM/TRM are not explained by the modest parameter increase alone: ablation studies by Jolicoeur-Martineau [1] show that parameter-matched modifications, such as doubling the number of backbone blocks or decoupling the **Solver** and **Generator** into separate networks (both at $\sim$10M parameters, a 100% increase), yield 7.9% and 4.9% worse performance on Sudoku Extreme respectively. This suggests that the gains are attributable to the architectural contribution of the **Reweighter** rather than the parameter increase.

Appendix C Formalization of SMC by the RIM framework

Sequential Monte Carlo Sequential Monte Carlo (SMC) are class of algorithms which allow to sample from a distribution sequentially. They were proposed as an improvement over the Importance Sampling (IS) [3, 4] which suffered from the weight degeneracy issues. In this section we will follow the description of the SCM from [5].

We want to approximate samples from a sequence $\gamma_1, \ldots, \gamma_T$ of probability distributions by defining the approximators $\hat{\gamma}_1, \ldots, \hat{\gamma}_T$ given unnormalized targets $\tilde{\gamma}_1, \ldots, \tilde{\gamma}_T$ and proposals $q_1, \ldots, q_T$. For each fixed $t \in \{1, \ldots, T\}$ we consider N samples. Then the output of the SMC sampler is a collection of weighted particles $\{(w_t^i, x_t^i)_{i=1}^N\}$.

Table C1: Parameter counts per model and benchmark.

Model	Sudoku	ARC-AGI 1&2	Maze
SimRIM/TRM	5,028,866	6,829,058	7,085,570
RIMA	5,554,178	7,354,370	7,348,226
RIMFormer	11,884,610	13,664,802	13,638,658

Table C2: Test accuracy on Sudoku Extreme across multiple runs.

Run	Test Accuracy
1	88.90
2	89.01
3	88.83
4	89.15
5	89.39

The particles evolve recursively by

$$x_t^i \sim \hat{\gamma}_{t-1}\left(x_{1:t-1}\right) q_t\left(x_t \mid x_{1:t-1}\right) \quad \text{and} \quad \tilde{w}_t^i = \frac{\tilde{\gamma}_t\left(x_{1:t}^i\right)}{\tilde{\gamma}_{t-1}\left(x_{1:t-1}^i\right) q_t\left(x_t \mid x_{1:t-1}\right)}.$$

Finally, we define $w_t^i = \frac{\tilde{w}_t^i}{\sum_{j=1}^N \tilde{w}_t^j}$ and approximate γ_t by

$$\hat{\gamma}_t\left(x_{1:t}\right) = \sum_{i=1}^N w_t^i \delta_{x_{1:t}^i}\left(x_{1:t}\right).$$

Now we can introduce a formal instantiation of the Sequential Monte Carlo by the RIM framework. Assume $t \in \{1, \ldots, T\}, i \in \{1, \ldots, N\}$, then a tuple $\langle G, S, \mathcal{R}\rangle$ describes an SMC-RIM, if its components are defined as

- The **Solver** S is given by $\hat{\gamma}_{t-1}\left(x_{1:t-1}\right) q_t\left(x_t | x_{1:t-1}\right)$ and samples $x_{1:t}^i$.
- The **Reweighter** $\mathcal{R} = (\mathcal{R}_L, \mathcal{R}_H)$ is given by

$$\mathcal{R}_L\left(x_{1:t}^i\right) = \tilde{w}_t^i = \frac{\tilde{\gamma}_t\left(x_{1:t}^i\right)}{\tilde{\gamma}_{t-1}\left(x_{1:t-1}^i\right) q_t\left(x_t^i | x_{1:t-1}\right)},$$
$$\text{and} \quad \mathcal{R}_H(x_{1:t}^i) = w_t^i = \frac{\mathcal{R}_L\left(x_{1:t}^i\right)}{\sum_{j=1}^N \mathcal{R}_L\left(x_{1:t}^j\right)}.$$

- The **Generator** G is given by $\hat{\gamma}_t$ delivering approximation of γ_t by

$$\hat{\gamma}_t = \sum\nolimits_{i=1}^N w_t^i \delta_{x_{1:t-1}^i}\left(x_{1:t-1}\right).$$

References

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

[2] Koch, F., Wever, M., Raisch, F., Tischler, B.: State-space models for tabular prior-data fitted networks. arXiv preprint arXiv:2510.14573 (2025)

[3] Robert, C.P., Casella, G., Casella, G.: Monte Carlo Statistical Methods vol. 2. Springer, ??? (2004)

[4] Kahn, H.: Random sampling (monte carlo) techniques in neutron attenuation problems. i. Nucleonics (US) Ceased publication **6**(See also NSA 3-990) (1950)

[5] Naesseth, C.A., Lindsten, F., Schön, T.B.: Elements of sequential monte carlo. Foundations and Trends® in Machine Learning **12**(3), 187–306 (2019)