Recursive Nonogram Reasoner – "NonoReason"

Inbal Ben Yehuda and Shany Danino
Technion - Israel Institute of Technology

Course: 046211 Deep Learning
Semester: Winter 2025-2026

January 2026

1 Introduction

1.1 Project Goal and Problem Statement

Nonograms are logic puzzles that consist of a blank grid with numbers along the top and left sides. These numbers act as clues indicating the length of consecutive black squares in that specific row or column. Human solution is achieved by using logical deductions, while cross-referencing the constraints to fill the grid correctly.

This kind of reasoning is a challenge to deep learning models, and classical algorithms require backtracking.

As of today, this kind of reasoning is a challenge to deep learning models, including Large Language Models (LLMs), which are yet unequipped to deal with certain types of tasks. That includes various kinds of visual puzzles, such as nonograms. A possible solution is called Tiny Recursive Model (TRM) [1], which was trained over other visual puzzles. The TRM smartly utilizes a small number of parameters to create a reasoning process that strives to both advance the solution and keep its correctness on each step. That emulates the general human process of solving such a puzzle.

In this project, we attempt to expand the existing TRM's framework to support solving nonograms, while preserving the principles of the model. One of tenet principles is the small number of parameters.

1.2 Previous Works

1.2.1 Algorithms and classical deep learning

Classical algorithms can be used to try and solve nonograms, with methods such as backtracking, line solving or genetic algorithms. These methods might struggle with computational complexity and use no organized method of "thinking".

Recently, Buades Rubio et al. [2] proposed a solution that combines heuristic algorithm with neural networks. This hybrid approach reduced computation time, with accuracy of 77.06% for grids of size 5X5, and 27.25% for grids of size 10X10.

1.2.2 Recursive Reasoning models

Wang et al. [3] proposed an Hierarchial Reasoning Model (HRM). This novel approach uses two layers of RNNs recursing at different frequencies - a low-level layer for fast execution of the task, and a high-level, slower layer for overall planning. The performance of the model is very good, but it requires complex training and a large number of parameters.

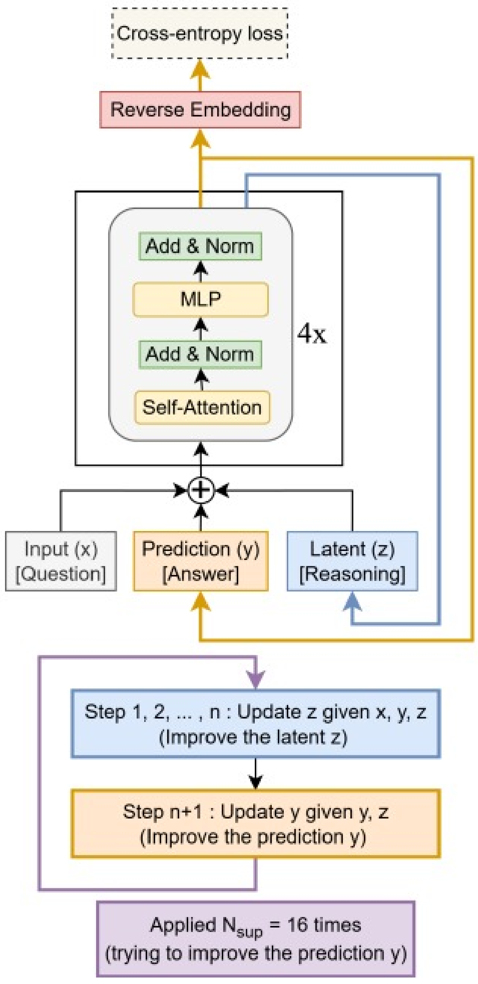
Based on this, Jolicoeur-Martineau [1] suggested the Tiny Recursive Model (TRM). It simplifies the architecture into a very compact network, achieving significantly higher generalization than HRM. Nonetheless, it preserves high performances over several tasks. The model is presented in detail in section 2.1.

2 Method

2.1 TRM original architecture

The foundation of our approach is the Tiny Recursive Model (TRM) [1], a "depth-in-time" architecture that replaces the traditional deep stack of transformer layers with a single, recurrently reused block. Unlike standard Transformers that process inputs in a single forward pass ("depth-in-space"), the TRM maintains a persistent hidden state ("Carry") consisting of a high-level reasoning vector (y) and a low-level grid for experimenting with different solutions (z).

In each iteration step, the model feeds its previous state back into itself, allowing it to iteratively refine its internal representation. The number of these steps is at most 16, but can be lowered according to the model's confidence in its internal representations. This mechanism is called Adaptive Computational Time (ACT).



Cross-entropy loss
Reverse Embedding
Add & Norm
MLP
Add & Norm
Self-Attention
4x
Input (x) [Question]
Prediction (y) [Answer]
Latent (z) [Reasoning]
Step 1, 2, ... , n : Update z given x, y, z (Improve the latent z)
Step n+1 : Update y given y, z (Improve the prediction y)
Applied N_{sup} = 16 times (trying to improve the prediction y)

Figure 1: Tiny Recursion Model (TRM) architecture

The core backbone utilizes a standard Transformer block with RMSNorm, SwiGLU activations, and Rotary Positional Embeddings, wrapped in a loop that cycles $H \times L$ times (defined by H_cycles and L_cycles), effectively mimicking a very deep network with a fraction of the parameters.

2.2 Architecture adjustments

The base architecture we adjusted was the one solving sudoku tasks using the attention mechanism. All adjustments presented here are compared to that model.

After all adjustments were made, the model's number of parameters remained very small: 7.7M parameters for nonograms of size 5x5, and 10.2M parameters for nonograms of size 10x10.

2.2.1 Separation of clues from the grid

While the input x for the sudoku task included only the starting sudoku board, our inputs were clues for the grid. Since the starting grid itself is completely empty, there was no need to feed it into the model.

2.2.2 Clues encoder

In order to feed the clues to the network naturally, we pass them through an encoder. It includes an MLP that takes the entire grid's constraints (row and column clues) as a flattened tensor of shape ($Size^2 \times 2 \times MaxClues$).

The MLP consists of two linear layers with a ReLU activation. This projects the larger, sparse constraint vector into a dense hidden dimension, creating a "grid embedding" that serves as the initial state for the reasoning loop.

2.2.3 Binary output head

We modified the final output layer to project the refined state to exactly 2 classes (Black/White pixel probabilities) instead of 9 classes (1-9, as in sudoku). That aligns the architecture with the binary requirements for the puzzle solution

2.2.4 Adaptive Computation Time (ACT) strategy

We found that the halting mechanism works differently depending on the puzzle difficulty. For simple 5×5 grids, the model successfully learned to stop early as soon as it solved the puzzle, which saved time. However, for harder 10×10 grids, allowing the model to decide when to stop led to a significant loss of stability, which in turn lowered the accuracy.

Therefore, we used a different approach for each size: for 5×5 tasks, we retained the ACT mechanism. For 10×10 tasks, we disabled it and forced the model to use all available reasoning steps ($N_{max} = 16$) to get the best possible result.

2.2.5 Loss

The original TRM implementation relied on stablemax cross entropy, which treats the output as a probability distribution over a vocabulary (in this case, 1-9). For nonograms, where every cell is either filled or empty, this approach isn't optimal. We replaced this function with a specialized binary cross entropy.

2.2.6 Hyper-parameters

In addition to all the architectural hard-coded changes, we also adjusted various of the model's parameters, aiming to optimize the performance over the nonogram task. We experimented with the learning rate, number of samples, epochs, batch size, weight decay, H_cycles and L_cycles. Plus, we experimented with the ACT mechanism; both with the probability to allow it to cause an early halt, and with disabling the mechanism completely.

3 Experiments and Results

3.1 Dataset Description

We used the nonogram dataset created by [2]. It's intended for developing and testing algorithms or machine learning models that can automatically solve nonogram puzzles. It includes puzzles of various dimensions:

- 5x5 - contains all possible combinations of 5X5 grids. Overall - 33,554,432 puzzles.

- 10x10 - 361,094 train grids, 15,274 test grids.
- 15x15 - 361,094 train grids, 15,274 test grids.

We used a sub-datasets of 5x5 and 10x10 puzzles, which contained a smaller number of samples. Each sub-dataset was split into 3 sets: Train, Test and Validation.

3.2 Experiments

We performed experiments over 2 grid sizes - 5x5 and 10x10. For each grid size we experimented with disablement of the ACT mechanism, and also with various sub-dataset sizes.

We will now focus on 4 main experiments:

1. Grids sized 5x5 with ACT mechanism
2. Grids sized 5x5 with no ACT mechanism
3. Grids sized 10x10 with ACT mechanism
4. Grids sized 10x10 with no ACT mechanism

In all cases, the batch size was 512 and the learning rate was $1e^{-4}$. The models were trained for 2,000 epochs for the 5x5 grids, and 1,500 epochs for the 10x10 grids. That is during to the constraints of the available resources.

Sizes of the sub-datasets for each experiments appear in Table 1.

The training sessions lasted an hour for the small sub-datasets of the 5x5 grids, and approximately 17 hours for the 10x10 grids. All training was done on a A100 GPU.

3.3 Results

We defined two types of accuracy:

1. Accuracy - the percentage of the correct decisions the model made when solving the nonogram. Correlates to the the accuracy of each pixel in the grid. This metric was used for internal understanding of the model's capabilities.

2. Exact Accuracy (EA) - the percentage of the nonograms that were 100% correct - meaning, every single pixel was correctly predicted by the model. This metric is the final criterion that was used when evaluating the model.

In order to get any nonogram to have an exact accuracy of 1, every single pixel in it should be predicted correctly, which requires a very high accuracy. A bigger grid has a bigger number of pixels, and even one mistake renders the whole nonogram's prediction incorrect. Because of that, we expected the results to be drastically different between experiments of different grid sizes.

3.4 Analysis

As shown in Table 1, the impact of the grid size over the final results is significant. The best result for the 5x5 nonograms was 85%, while 10x10 nonograms achieved at most 51.2%.

The effect of the ACT mechanism over the model's behavior is presented in Figure 2. In the case of a 5x5 nonogram, the mechanism quickly strives to train over a small amount of steps - around 3. When working with a 10x10 grid, the mechanism first tries

Exp.	**Grid Size**	**ACT**	**Train Set Size**	**Test Set Size**	**Train EA (%)**	**Test EA (%)**
1.a	5x5	+	5,000	1,000	99.4	79.6
1.b	5x5	+	50,000	10,000	99.8	**85**
2	5x5	-	5,000	1,000	93.7	79.8
3	10x10	+	50,000	10,000	82.9	14.7
4	10x10	-	50,000	10,000	68.3	**51.2**

Table 1: Overall Performance of the model for each experiment

to reduce the number of steps, but the model becomes very unstable. During the entire run, the mechanism doesn't manage to stabilize on a small number of steps, and is overall unstable. This behavior lead us to explore the option of disabling the ACT altogether, which indeed achieved much improved results. The ACT's impact was nearly nonexistent in the smaller grids, yet 10x10 nonograms received a dramatic boost in results when ACT was disabled - jumping from 14.7% to 51.2%.

In all cases, the larger Train set helped with overfitting, and achieved better results.

3.5 Comparing Results

Our main baseline for the task of solving nonograms came from [2], as mentioned in section 1.

The comparison is shown in Table 2. Our results are demonstrably better than those of the other model, with an improvement of 7.9% for 5x5 nonograms, and 12% for 10x10 nonograms. The fact that the larger difference is present in the larger grids might suggest that the TRM is better equipped to deal with big grids.

Model	**Grid Size**	**Exact Accuracy (%)**
Heuristic + Neural Network	5x5	77.1
	10x10	39.2
Our	5x5	85
	10x10	51.2
Original TRM - sudoku	9x9	74.7

Table 2: Performance of the model for each experiment

Another source of comparison was the Gemini 3 Pro model. That model was given over 20 different nonograms of size 10x10, and its exact accuracy was measured. While highly dependent on the exact prompt given, we observed that the model fixated either on solving the rows clues or the column clues, rarely looking at both at the same time. That caused it to have an exact accuracy of 0%.

We also compared the results of our model to these of the original TRM model's sudoku task. The results are shown in Table 2. These results demonstrate that, even though our model was able to adjust to the nonogram task, it wasn't optimized and trained to its full capabilities.

We believe a big part of the difference lies in both the training length and the volume of training samples. While we trained the 10x10 grids for 1,400 epochs and over 50,000 examples, the original TRM was trained over 1,000,000 examples (1,000 examples, with 1,000 augmentations on each one) for 50,000 epochs. Plus, the number of pixels in a sudoku is 81, compared to our 100. We already saw that the grid's size has a large impact over the results.

The results suggest the TRM was successfully adjusted to work with nonograms.

4 Conclusion and Future Work

4.1 Summary of Contributions

In this work, we successfully adapted the Tiny Recursive Model (TRM) to solve nonogram puzzles, extending the architecture's capabilities from its original Sudoku-solving baseline to the domain of strict binary constraints. Our primary contribution is the architectural transformation of the TRM: we replaced the standard inputs with a domain-specific clue encoder and substituted the original multi-class output head with a binary head. These adjustments created a highly efficient model capable of solving logic puzzles that even massive Large Language Models (LLMs) find difficult. Importantly, we kept the tiny nature of the model, yet even with limited computing power, our model achieved accuracy levels comparable to the key benchmarks in this report.

4.2 Limitations

Despite the impressive results achieved by our model, several challenges and open questions remain. First, the Adaptive Computation Time (ACT) mechanism proved unstable for complex tasks. We disabled the model's ability to self-terminate on 10×10 grids, relying instead on a fixed, computationally expensive loop of 16 steps to ensure accuracy. Second, the model required about 50,000 training examples to achieve good results on 10×10 grids. Finally, our architecture was optimized and validated on 5×5 and 10×10 grids. Extending the architecture to 15×15 or larger puzzles remains a significant challenge, as the combinatorial search space grows exponentially with size.

4.3 Future Directions

To address the limitations outlined above and further explore the capabilities of recursive reasoning for nonograms, we propose several directions for future research. First, future work should focus on resolving the instability of the ACT mechanism on 10×10 grids. We believe that separating the mechanism's learning metrics from the model's could lead to significant improvements, an investigation that we could not fully explore in this project due to time and resource constraints.

In addition, future efforts should aim to improve the model's sample efficiency. Given the dataset currently required ($\sim$50,000 samples), investigating techniques to generalize effectively from significantly fewer training examples could yield a more stable model with reduced overfitting

Finally, extending the architecture to larger puzzle sizes, such as 15×15, is a natural next step. Future research can focus on determining whether the current recursive depth is sufficient for these larger tasks, or if the number of reasoning steps must scale up to maintain accuracy.

References

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

[2] Rubio, José María Buades, et al., “Solving nonograms using neural networks,” *Entertainment Computing 50 (2024): 100652.*

[3] Wang, Guan, et al., “Hierarchical Reasoning Model,” *arXiv preprint arXiv:2506.21734*, (2025).

5 Ethics Statement

5.1 Introduction

Students names: Inbal Ben Yehuda and Shany Danino
Project Title: Recursive Nonogram Reasoner – "NonoReason"
Project Description: The goal of this project is to develop an AI model capable of solving nonograms - logic puzzles defined by row and column constraints. Based on recursive reasoning, the model mimics human problem-solving by filling the grid step-by-step, constantly cross-referencing constraints to verify the solution.

5.2 Gemini’s answers

1. **List of stakeholders**
   - The AI Research Community (Computer Science): Interested in the algorithmic breakthrough of using small, recursive models for logic instead of massive LLMs.
   - Cognitive Scientists & Psychologists: Interested in the model’s ability to mimic human-like step-by-step deductive reasoning.
   - Puzzle Enthusiasts (End Users): Interested in a tool that can teach them logic and provide hints without spoiling the game.
2. **Explanations**
   - For the AI Research Community: A technical breakdown demonstrating how recursive reasoning achieves high accuracy on logic tasks with minimal data and compute, challenging the field’s reliance on massive LLMs.
   - For Cognitive Scientists & Psychologists: A descriptive overview explaining how the algorithm mimics human deduction—specifically step-by-step constraint checking—to serve as a computational model for cognitive processes.
   - For Puzzle Enthusiasts (End Users): A simple guide presenting the AI as a "smart tutor" that explains the logic behind each move to help players learn strategies, rather than just revealing the final solution.
3. **Responsibility for the explanations**
   The Project Development Team is responsible for tailoring and delivering these explanations to all three groups. As the creators of the architecture, only the team

members possess the deep technical understanding required to write the academic papers for the Research Community. Simultaneously, they must "translate" this technical knowledge into accessible user interface text for End Users and comparative psychological studies for Cognitive Scientists. Effective communication is a core part of the project's success, ensuring the technology is understood by those who study it and those who use it.

5.3 Our reflection

In order to make Gemini's responses more ethical, we believe a few additions are necessary. First, we agree that it is our responsibility to deliver clear and comprehensive explanations to each stakeholder. Regarding the AI research community, alongside the technical details critical for future research, we should add an ethical note emphasizing the importance of using this tool responsibly. For cognitive scientists, in addition to accessible instructions on how to use the model, we must explain the importance of not testing human subjects against the model or comparing humans directly to it. Finally, for all stakeholders, it is crucial to emphasize the model's limitations, acknowledging that it is not perfect and cannot be fully relied upon. We must clarify that the model only attempts to simulate specific principles of human logic, but it does not truly mimic human cognition or 'think' like a person.