Data-Efficient Learning for Constraint Satisfaction Problems via Relational Biases and Hard Axiom Clamping

Enqiang Zhu[1] **Ke Wang**[1] **Yu Zhang**[2] **Shengzhi Wang**[3] **Xianhang Luo**[4] **Chanjuan Liu**[3*]
[1]Institute of Computing Science and Technology, Guangzhou University, Guangzhou, China
[2]School of Computer Science, Peking University, Beijing, China
[3]School of Computer Science and Technology, Dalian University of Technology, Dalian, China
[4] School of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan, China
zhuenqiang@gzhu.edu.cn kewang@e.gzhu.edu.cn yuzhang.cs@stu.pku.edu.cn
wangshengzhi@mail.dlut.edu.cn luoxianhang2022@gmail.com chanjuanliu@dlut.edu.cn

Abstract

Constraint satisfaction problems (CSPs) serve as critical benchmarks for neural algorithmic reasoning, requiring accurate multi-step constraint propagation to determine variable values that satisfy all specified constraints. Existing neural solvers often rely on extensive training datasets, struggle with distribution shifts, or allow fixed clues to drift in continuous spaces. We introduce CSP-RSB, a data-efficient recurrent Transformer that directly incorporates constraint topology into the attention mechanism through a learnable relational bias. We also employ a hard axiom-clamping strategy to fix clues after each recurrence-constraint projection and use QK-Norm to stabilize a two-layer block. On 9×9 Sudoku, CSP-RSB achieves 99.94% board accuracy on the RRN-test, 99.13% on unfiltered Kaggle, and 99.78% on filtered Kaggle, surpassing the previous best, Trial & Error 2025, which reported 99.40%, 98.90%, and 99.50%, respectively. Furthermore, it successfully solves Erdős–Rényi graph coloring with 100% accuracy in 9 epochs and maintains 99.5-100% accuracy on 10×10 binary puzzles. By incorporating explicit topological structure, CSP-RSB reduces the depth and data required for effective algorithmic reasoning.

1 Introduction

Constraint satisfaction problems (CSPs) require assigning values to variables from finite domains such that all constraints are satisfied. Tasks such as Sudoku, binary puzzles, and graph coloring naturally fit into this framework. What makes CSPs hard is the need for global consistency: a single wrong assignment invalidates the entire solution. CSPs thus serve as a testbed for neural algorithmic reasoning, evaluating whether a model can perform multi-step constraint propagation. Despite growing interest in the reasoning abilities of large language models, these models often hallucinate, violate constraints, and fail under distribution shifts [1–5]. While they can produce plausible intermediate text, they often fail to maintain a consistent latent state across extended chains of constraints. CSPs eliminate linguistic ambiguity and raise a fundamental question: can a learned model propagate constraints until reaching a valid fixed point? Standard self-attention enables global communication but does not inherently encode which relations (rows, columns, blocks, edges) govern constraint propagation [6–8].

Existing neural CSP solvers use recurrence and relational structure but encounter an architectural gap. Recurrent Relational Networks (RRNs) rely on hand-crafted message passing whose structural priors

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

must be re-engineered per task and degrade under distribution shift [9]. Recurrent Transformers implement tied attention for iterative updates but lack structural inductive biases, requiring large datasets to learn the topology from scratch [10]. Fixed clues can drift across layers within the continuous state space. Recent causal language and trial-and-error methods [3, 11] do not address end-to-end representation learning. Instead, they rely on costly test-time search or external supervision rather than producing a global solution in a single forward pass. For further details on related work, please refer to Appendix A.

In this paper, we ask whether explicit structural priors can replace depth and data in forward-only CSP solving. We propose CSP-RSB, a topology-aware recurrent Transformer that mitigates data inefficiency and clue drift. The model combines additive relational bias, hard axiom clamping, and QK-Norm in a two-layer recurrent Transformer shared across 16 steps. We study their integration and data-efficiency trade-offs. Relation types and adjacency are supplied by the task, while their weights are learned. Our main contributions are as follows.

(1) Topology-aware relational bias. We integrate CSP relations as learnable biases on attention logits, together with structural and fixed-mutable embeddings. This approach enables the recurrent block to focus on learning the update rule rather than inferring the constraint graph from data. Ablation studies show that this routing is required for board-level reasoning and accelerates convergence; see Table 2.

(2) Hard axiom clamping strategy. Fixed clues are treated as axioms rather than ordinary tokens. CSP-RSB resets the hidden state at clue positions to its initial clue embedding after every recurrent step and trains only on the mutable cells. This clamping operation is most useful in low-data regimes (gain of +25.7 pts at 5% data, +6.3 pts at 10%).

(3) Board accuracy and dataset. In 9×9 Sudoku, the CSP-RSB model trained on the RRN-180k dataset reaches 99.94% accuracy on the RRN-test Sudoku dataset, 99.78% on the filtered Kaggle split, and 99.13% on the unfiltered Kaggle split, outperforms the previous best (Trial & Error) baseline, and significantly surpasses the performance of retrained recurrent baselines. In addition, we develop a new dataset, Given-49k, which consists of a 49,151-puzzle Sudoku subset with all naked single-solvable boards removed. This dataset is designed as a more challenging, low-data testbed for deductive reasoning beyond simplistic heuristics.

(4) General evidence for structural priors. We conducted a series of experiments to assess data efficiency, distribution shifts, recurrence depth, stability, ablation studies, reasoning dynamics, and topology transfer. By modifying the relation set, the same model reaches 100% accuracy in coloring Erdős–Rényi graphs in 9 epochs, and reaches 99.5–100% on 10×10 binary puzzles. Incorporating explicit structural information reduces the depth and data required to match prior recurrent solvers.

2 Preliminaries

A CSP is represented as a triple $< V, D, C >$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a finite set of n variables, $D = \{D_1, D_2, \ldots, D_n\}$ is a family of finite sets for which each D_i is the domain of possible values for v_i, and $C = \{c_1, c_2, \ldots, c_m\}$ is a set of constraints. Given a variable v_i and a value $d \in D_i$, the expression $v_i = d$ is called an *atom*, indicating that the variable v_i is assigned the value d. A *constraint* $c \in C$ is defined over a sequence of variables $(v_i, v_{i+1}, \ldots, v_j)$ and acts as a function $c : D_i \times \cdots \times D_j \to \{\text{TRUE}, \text{FALSE}\}$. Here, $c(< d_i, \ldots, d_j >) = \text{TRUE}$ signifies that the combination of atoms $v_k = d_k$ for $k = i, \ldots, j$ can coexist concurrently. Conversely, if the constraint evaluates to FALSE, then that combination cannot hold simultaneously. An *evaluation* is a set of n atoms $\{x_k = d_k | k = 1, 2, \ldots, n, d_i \in D_i\}$. An evaluation becomes a *solution* if the expression $c(< d_1, \ldots, d_n >)$ evaluates to TRUE for each $c \in C$.

A commonly used type is the *cardinality constraint*. Let b and u be nonnegative integers such that $b \leq u$. Given variables $x_i, x_{i+1}, \ldots, x_j$, $i \leq j$, and a corresponding group of reference values $a_i, \ldots, a_j$ such that $a_k \in D_k$, $k = i, i+1, \ldots, j$, for each input $< d_i, \ldots, d_j > \in D_i \times \cdots \times D_j$, the cardinality constraint σ on the variables $x_i, x_{i+1}, \ldots, x_j$ is defined as follows:

$$\sigma(< d_i, \ldots, d_j >) = \begin{cases} \text{TRUE}, & \text{if } b \leq \left|\{d_i = a_i, \ldots, d_j = a_j\}\right| \leq u, \\ \text{FALSE}, & \text{otherwise.} \end{cases}$$

Here, $d_k = a_k$ is true if a_k and d_k have the same value, and false otherwise; the vertical bars indicate the number of true items. In essence, σ requires that the number of variables assigned values equal to their corresponding reference values falls within the inclusive range between b and u. When $b = u$, the constraint simplifies to

$$\sigma(< d_i, \ldots, d_j >) = \begin{cases} \text{TRUE}, & \text{if } \big|\{d_i = a_i, \ldots, d_j = a_j\}\big| = u, \\ \text{FALSE}, & \text{otherwise.} \end{cases}$$

If further $i = j$ and $u = 1$, we obtain the unary constraint $\sigma(< d_i >) = \text{TRUE}$ if $d_i = a_i$, and $\sigma(< d_i >) = \text{FALSE}$ otherwise.

CSP for Sudoku. Let G be an $n \times n$ grid whose n^2 cells are indexed by $p = (i-1)n + j$ for rows i and columns j in $\{1, \ldots, n\}$, where $n = k^2$ is a perfect square. An $n \times n$ Sudoku puzzle on G involves filling each cell with a number from the set $\{1, \ldots, n\}$ in such a way that each number appears exactly once in every row, every column, and within each of the n distinct $k \times k$ block. This problem has been shown to be NP-complete for general values of n [12]. The puzzle is modeled as a CSP $< V, D, C >$ as follows: The variable set $V = \{\text{cell}_1, \ldots, \text{cell}_{n^2}\}$ represents the cells of G. Each domain D_p is defined as the set $\{1, \ldots, n\}$, indicating that all cells share the same set of possible values. Each input token $x_p \in \{0\} \cup \{1, \ldots, n\}$ signifies the initial state of cell p, where 0 denotes an empty mutable cell, while any nonzero value represents a fixed clue.

The constraints fall into two types. First, for each pre-filled cell p with a specified digit $d \in \{1, 2, \ldots, n\}$, we impose a unary constraint $\text{cell}_p = d$, thus fixing the variable to that specific value. Second, for every digit d and for every subset $S \subseteq \{1, 2, \ldots, n^2\}$ of exactly n cell indices that comprise a complete row, complete column, or a block, we apply a cardinality constraint defined as

$$\big|\{\text{cell}_\ell = d \mid \ell \in S\}\big| = 1,$$

which states that exactly one cell within that row, column, or block is assigned the value d. A solution to this CSP corresponds precisely to a completed Sudoku grid that adheres to the initial clues.

3 Methodology

This section details the proposed CSP-RSB model. CSP-RSB combines four components: a topology-aware relational bias, structural embeddings, a hard axiom clamping mechanism for recurrent constraint propagation, and QK-Norm stabilization. Appendix B, Figure 4 provides an overview. The following subsections introduce each technique in depth.

3.1 Topology-aware relational bias and structural embedding

Relational bias. We incorporate topological priors directly into attention logits via a learnable relational bias, inspired by relation-aware self-attention [13] and additive positional biases [14]. Let $\Psi \in \mathbb{R}^{L \times L}$ denote the bias between cell tokens. For each relation type k in the topology, we define an edge set $\text{rel}_k \subseteq \{1, \ldots, L\}^2$. The bias is

$$\Psi_{i,j} = \sum_{k \in \mathcal{R}} w_k \cdot \mathbf{1}\{(i,j) \in \text{rel}_k\}, \tag{1}$$

where w_k is learnable, and $\mathbf{1}\{\cdot\}$ is the indicator function. Unrelated pairs receive zero bias, so global communication is unaffected. When a `CLS` token is used, we extend to $\tilde{\Psi} \in \mathbb{R}^{(L+1)\times(L+1)}$ by setting $\tilde{\Psi}_{i,j} = \Psi_{i,j}$ on cell-token pairs and $\tilde{\Psi}_{0,j} = \tilde{\Psi}_{i,0} = w_{\text{CLS}}$ on the `CLS` row and column, with w_{CLS} a learned scalar. The same relation weights are shared across all T recurrent steps, reusing a single soft logic field. For Sudoku, $\mathcal{R} = \{\text{row}, \text{col}, \text{blk}\}$ and self-loops are excluded so that the bias between any two distinct cells i and j gets a learned positive weight whenever the two cells share a row, column, or block.

$$\Psi^{\text{Sudoku}}_{i,j} = w_{\text{row}} \cdot \mathbf{1}\{r_i = r_j,\, i \neq j\} + w_{\text{col}} \cdot \mathbf{1}\{c_i = c_j,\, i \neq j\} + w_{\text{blk}} \cdot \mathbf{1}\{b_i = b_j,\, i \neq j\}.$$

Figure 1 compares the cell-cell attention before training. Each of the L=81 Sudoku cells attends to 80 other cells, 20 of which are topological neighbors. With Ψ=$\mathbf{0}$ (panel a), $\text{softmax}(\mathbf{0})$ gives every key equal weight $1/80$, so each query sends 25% of its attention to its 20 neighbors. Panel b

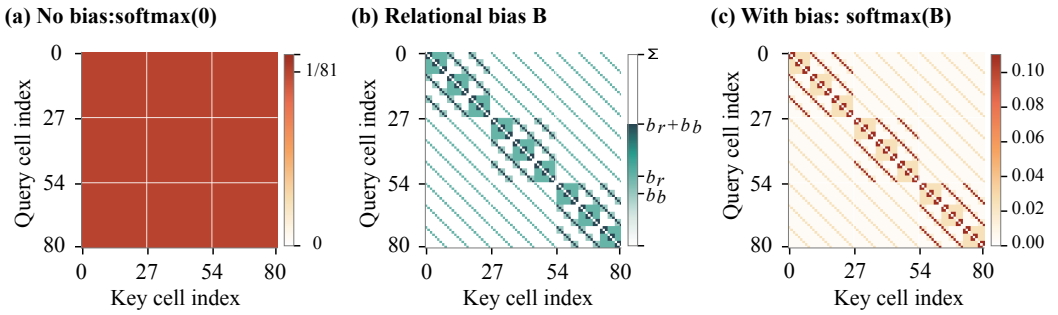

Figure 1: Relational bias at initialization (no parameters trained). (a) $\mathrm{softmax}(\mathbf{0})$: every query distributes its attention uniformly, $1/80$ per key, so the 20 topological neighbors collectively receive 25%. (b) The bias matrix Ψ, with one learnable scalar w_k per relation type. (c) $\mathrm{softmax}(\Psi)$: each query's attention concentrates on its 20 neighbors, while the remaining 60 entries are reduced but non-zero.

shows Ψ itself; panel c shows $\mathrm{softmax}(\Psi)$, which concentrates the attention of every query on its 20 neighbors and reduces, but does not zero, the remaining entries. After training, the bias-on/off accuracy gap shrinks from -20.4 pts at 10% data to -0.3 pts at 100% (Appendix F): the bias reduces the data needed to reach a given accuracy without raising the asymptote.

Structural embedding. Traditional Transformers [6] generally employ 1D structural embedding, which is inadequate for the multi-dimensional structure inherent in many CSPs. CSP-RSB addresses this limitation by initializing the cell token sequence using a composite embedding layer that combines structural and functional information. For any variable v_i, its initial cell representation is formulated as follows:

$$E_i = E_{val}(v_i) + E_{pos}(i) + E_{type}(\tau_i), \tag{2}$$

where E_{val} maps the numerical token values into a d_{model}-dimensional latent space (which is distinct from the domain D. E_{pos} is the structural embedding, encoding the topological coordinates of the variable within the constraint graph or grid. E_{type} is the type embedding, where $\tau_i \in \{\text{fixed, mutable}\}$ indicates whether variable v_i is a predefined clue (axiom) or an unknown variable that needs to be inferred. For Sudoku, the embedding E_{pos} can be decomposed into orthogonal spatial dimensions $E_{row}(r_i), E_{col}(c_i), E_{blk}(b_i)$, and

$$E_i^{\text{Sudoku}} = E_{val}(v_i) + E_{row}(r_i) + E_{col}(c_i) + E_{blk}(b_i) + E_{type}(\tau_i)$$

Here, r_i, c_i, and b_i represent the row, column, and block indices of cell i. Each cell, therefore, lives in three orthogonal index spaces (row, column, block), so the same parameter does not have to encode all three at once.

3.2 Hard axiom clamping for recurrent constraint propagation

We adopt a weight-tied recurrent architecture inspired by the Universal Transformer [15] for constraint reasoning. The initial latent state $H^{(0)}$ is defined as

$$H^{(0)} = \mathrm{Norm}([e_{\mathrm{CLS}}; E_1; \ldots; E_L]) \in \mathbb{R}^{(L+1)\times d_{\text{model}}} \tag{3}$$

where e_{CLS} denotes a learned global token. In Section 4 (Table 2) and Appendix C, we show in CSP-RSB that this token is not required; the relational priors already carry the global information that a `CLS` token would typically summarize. In the no-`CLS` ablation scenario, $H^{(0)}$ consists of the sequence of L cell tokens.

The unprojected block output at step $t+1$ is produced by the weight-tied operator, which consumes the projected state of the previous step:

$$H^{(t+1)} = \mathcal{F}(\tilde{H}^{(t)}, \tilde{\Psi}; \theta), \quad t \in \{0, 1, \ldots, T-1\}, \tag{4}$$

with $\tilde{H}^{(0)} := H^{(0)}$. Here $\mathcal{F}$ is a two-layer Transformer block with parameters θ shared across all T steps. The matrix $\tilde{\Psi}$ is a learnable relational bias associated with the current set of tokens, which

includes both cell tokens and the CLS token. Formula (4) forces the network to learn a single reasoning operator that is uniformly applied over time. Empirically, this prevents the late-training accuracy degradation observed in deep non-recurrent stacks on challenging Sudoku puzzles (Appendix I)

Without explicit anchoring, the hidden representations at fixed clue positions drift across recurrent steps, a phenomenon we term *axiom drift*. This drifted state then propagates into mutable cells through attention. To prevent this, we separate the optimization space (mutable variables) from the constraint space (fixed axioms) via a constraint projection applied after every step $t+1$:

$$\tilde{H}^{(t+1)} = M \odot H^{(t+1)} + (1 - M) \odot H^{(0)} \tag{5}$$

where $H^{(t+1)}$, $\tilde{H}^{(t+1)} \in \mathbb{R}^{(L+1)\times d_{\text{model}}}$, and $M \in \{0,1\}^{L+1}$ is a binary mask with M_i=1 at mutable cells and at the CLS index, and M_i=0 at given clues. We apply this constraint projection (Formula 5) at every recurrent step, so mutable cells always attend to clue positions whose hidden state matches the input clue. Gradients only flow through mutable cells, which prevents the model from rewriting input axioms. The clamp adds 25.7 pts on RRN-5% and 6.3 pts on RRN-10% (Appendix H.1).

3.3 Training objective and stabilization

The recurrent block uses relational multi-head attention (R-MHA). To prevent softmax saturation under repeated application, we apply LayerNorm [16] to Q and K before the dot product, following the QK-Norm method of [17]. With relational bias $\tilde{\Psi}$, the attention is

$$\text{Attn}(Q,K,V) = \text{softmax}\left(\frac{\text{LN}(Q)\text{LN}(K)^T}{\sqrt{d_h}} + \tilde{\Psi}\right)V, \tag{6}$$

where d_h is the per-head dimension. CSP-RSB is supervised at every recurrent step. Let $z^{(t)} \in \mathbb{R}^{L\times|\mathcal{D}|}$ denote per-cell logits from the value head on $\tilde{H}^{(t)}$ (the CLS row is dropped) and y the ground truth. The per-step loss is the cross-entropy on mutable cells:

$$\mathcal{L}_{SFT}^{(t)} = -\frac{1}{|\{i : M_i = 1\}|}\sum_{i:\, M_i=1} \log \text{softmax}\Big(z_i^{(t)}\Big)_{y_i}, \tag{7}$$

and the total objective sums these losses with linear weights:

$$\mathcal{L}_{total} = \sum_{t=1}^{T} w_t \cdot \mathcal{L}_{SFT}^{(t)}, \qquad w_t = \tfrac{t}{T}. \tag{8}$$

The schedule $w_t = t/T$ down-weights early exploratory steps and puts more weight on the final step. Further rationale for the use of QK-Norm and the linear weighting can be found in Appendix D.

4 Experiments

4.1 Datasets and baselines

Datasets. We train CSP-RSB on two Sudoku training sets that share the same evaluation pipeline. The first is the standard RRN-180k benchmark [9], containing 180k training, 18k validation, and 18k test puzzles, each with a unique solution. The second, Given-49k, is a harder subset derived from the RRN-180k training split. We repeatedly apply the naked single heuristic to the original training pool; puzzles fully solved by this rule are discarded as trivial, while those with remaining empty cells are kept as challenging instances. The final set comprises 49,151 uniquely solvable training puzzles (48,138 with 17 clues, 1,013 with 19–33 clues). The validation and test splits are unchanged from RRN-180k. Appendix E provides the construction pipeline, clue-count distribution, and filtering schematic. For all tests, we report board accuracy (Acc), which requires that every mutable cell be correctly predicted. To evaluate out-of-distribution (OOD) robustness and immunity to logic drift, we perform zero-shot evaluation on the public Kaggle Sudoku dump in the two forms adopted by prior work [3, 11]: the *unfiltered* version (full 3M puzzles from [18]) and the *filtered* version (1.9M puzzles solvable by seven human strategies, curated by [3]). The unfiltered split is the harder evaluation, as it includes puzzles outside the strategy-solvable region. No model is trained on Kaggle data.

Baselines. We compare CSP-RSB against four state-of-the-art CSP solvers. Three have public implementations: RRN [9], Recurrent TR [10], and Causal LM [3]. We re-evaluated these baselines

using their official codebases under our pipeline. The fourth, Trial & Error [11], does not provide executable code, so we report its published results. Our models trained on RRN-180k and Given-49k are denoted CSP-RSB-180 and CSP-RSB-49, respectively. In all experiments, CSP-RSB uses 16 recurrent steps for both training and evaluation, consistent with baseline practices. Appendix E gives further implementation details. We additionally include OR-Tools CP-SAT [19], an exact algorithm that serves as a non-learning reference by using explicit Sudoku constraints and no training data. It achieves 100% board accuracy on the RRN Test, Kaggle Unfiltered, and Kaggle Filtered datasets, as shown in Table 1 and further detailed in Appendix W.

4.2 Comparison of board accuracy

Table 1 presents the board accuracy achieved by all compared methods. CSP-RSB-49 achieves 99.84% on the RRN test set, 97.81% on the unfiltered Kaggle split, and 99.27% on the filtered Kaggle split, +12 to +17 pts above the strongest retrained baseline Recurrent TR. CSP-RSB-180 reaches 99.94% on the RRN test set, above the previous best (Trial & Error, 99.40%). In the Kaggle zero-shot benchmark, CSP-RSB-180 records 99.78% on the filtered split and 99.13% on the unfiltered split. Both Kaggle splits sit above Trial & Error (99.50% and 98.90%, respectively) and 18–20 pts above the retrained recurrent baselines on the unfiltered split (RRN 78.99 %, Recurrent TR 82.87 %, Causal LM 81.12 %). While the unfiltered split poses the most significant challenge for prior recurrent solvers (82.87%), CSP-RSB-180 stays at 99.13% on this split with the same forward-only inference used for the in-distribution split. Although Trial & Error matches us within 0.5 pt on the filtered Kaggle split, it relies on autoregressive transcripts, backtracking, and depth-first search, which can lead to scalability and computational efficiency issues. Observe that CSP-RSB-49 uses only 49k puzzles (approximately a quarter of the data used by CSP-RSB-180) yet achieves accuracy comparable to that of CSP-RSB-180 across all test sets. The following section further investigates this property with a dedicated experiment. CP-SAT serves as a separate non-learning reference and achieves 100% board accuracy on all three test sets by using explicit constraints and search, as detailed in Appendix W. In addition, measured CSP-RSB inference costs and analytical Trial & Error costs are reported separately in Appendix V.

Table 1: Sudoku board accuracy comparison. All learned models were trained until convergence. CSP-RSB results are means over three seeds (full mean±std in Appendix P). RRN, Recurrent TR, and Causal LM were retrained under matched parameters and budgets (see Appendix O, "Baselines" paragraph). We use the Trial & Error numbers from the original paper [11] and report them only for RRN-180k, since the original work does not evaluate on Given-49k. OR-Tools CP-SAT is an exact, non-learning reference requiring no training data.

Training Data	Model	Acc (RRN Test)	Acc (Kaggle Unfiltered)	Acc (Kaggle Filtered)
RRN-180k	RRN (2018) [9]	96.62%	78.99%	91.06%
	Recurrent TR (2023) [10]	99.51%	82.87%	95.25%
	Causal LM (Shah et al., 2024) [3]	95.96%	81.12%	93.17%
	Trial & Error (2025; test-time search) [11]	99.40%	98.90%	99.50%
	CSP-RSB-180 (Ours)	**99.94%**	**99.13%**	**99.78%**
Given-49k	RRN (2018) [9]	82.94%	72.03%	78.98%
	Recurrent TR (2023) [10]	85.68%	79.21%	86.36%
	Causal LM (Shah et al., 2024) [3]	77.01%	68.40%	74.51%
	CSP-RSB-49 (Ours)	**99.84%**	**97.81%**	**99.27%**
No training	OR-Tools CP-SAT	100.00%	100.00%	100.00%

4.3 Data efficiency analysis

We evaluate CSP-RSB using fractional subsets of RRN-180k, as detailed in Appendix F (Table 3). With just 10% of the data (18k samples), the model achieves 84.10% accuracy. At 30% (54k samples), accuracy rises to 99.32%, above the 96.62% reached by RRN trained on the full 180k corpus. At 50% (90k samples), it reaches 99.71% accuracy, above Trial & Error's 99.40%. The full 180k corpus reaches 99.94%. Internalizing the topological structure reduces sample complexity by an order of magnitude.

To confirm that the observed efficiency stems from relational bias rather than the Transformer's generic capacity, we ablate the learnable bias at each data ratio. Figure 2 shows the training trajectories, while Appendix F (Table 3) lists best-epoch accuracies side by side, with two derived columns

(error reduction and relative error ratio) that remain informative near saturation. The bias also shows a distinct initialization-time signature, directing attention toward legal neighbors before training (Appendix G). Eliminating the bias lowers accuracy at every ratio: from 84.10% to 63.68% at 10% data, from 99.32% to 93.26% at 30%, from 99.71% to 98.46% at 50%, and from 99.94% to 99.59% at 100%. First, the performance gap narrows as the amount of data increases, with differences of –20.4, –6.1, –1.2, and –0.3 points, respectively. This indicates a reduction in sample complexity rather than a change in asymptotic accuracy; the bias-free model achieves the same ceiling but requires roughly an order of magnitude more data. Second, at the low-data extreme of 10%, the bias distinguishes a partially solved board (63.68%) from one that is nearly saturated (84.10%). At full data, the bias contribution is only 0.3 points. In other words, the relational prior mostly improves sample efficiency rather than final accuracy.

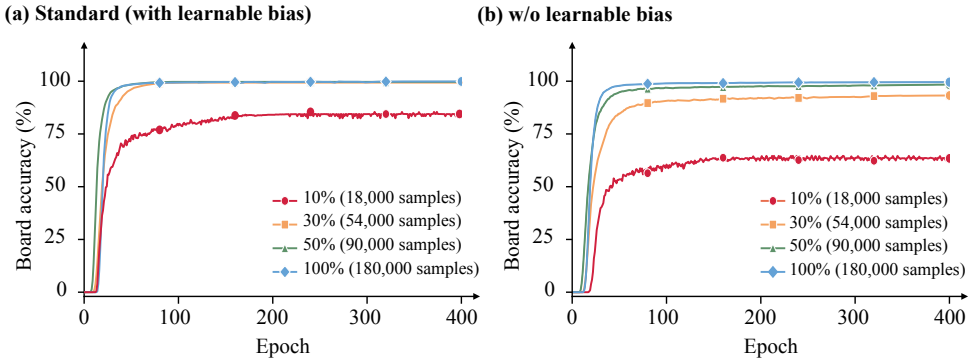

Figure 2: Validation trajectories on RRN-180k subsets: with versus without learnable relational bias. Left (a): The Standard configuration (with bias) saturates within a few epochs at data levels of 30%, 50%, and 100%, achieving approximately 84% accuracy at 10%. Right (b): The absence of bias results in higher overall accuracy trajectories remaining beneath those of the bias-enabled counterparts; the 30% run peaks around 93%, while the 10% run stabilizes near 64%. The influence of bias is most pronounced when data availability is limited.

4.4 Constraint projection analysis

We conduct an ablation study of the constraint projection module under low-data training conditions, using 5% and 2.5% subsets of RRN-180k (9,000 and 4,500 samples, respectively). These subsets are chosen because the embeddings already distinguish between fixed and mutable cells at the input level, allowing the network to partially compensate for the absence of the projection. To isolate the architectural prior, we implement a quarter-scale configuration (Lt-$\frac{1}{4}$; see Appendix O, Table 17). An analysis using a different setup that omits the type embedding while using the full-capacity backbone is reported in Appendix H.1.

Removing projection lowers cell accuracy from 80.59% to 58.47% with 5% of the data, and from 70.14% to 40.12% with 2.5% of the data (Figure 3; Appendix H). Without the clamp, the hidden states at the clue positions are not reset at each step, forcing the network to re-encode clues from scratch. So, projection yields gains of 22.1 and 30.0 percentage points; the gap grows as the training set shrinks. Figure 3 also tracks per-epoch validation trajectories and an internal violation rate, the fraction of clue positions where the value head on the unprojected output $H^{(t+1)}$ reads off a digit different from the input clue. A similar reduction in sample complexity appears in board accuracy when we remove the type embedding from the full backbone: the projection gap increases from +6.3 points at 10% data to +25.7 points at 5% (see Appendix H.1).

4.5 Component analysis

Table 2 evaluates each structural component on the data-rich Given-49k dataset and the low-data RRN-18k split. The latter is the 10% subset of RRN-180k (18,000 puzzles) and is also used as the 0.10 ratio in Table 3 of Appendix F.

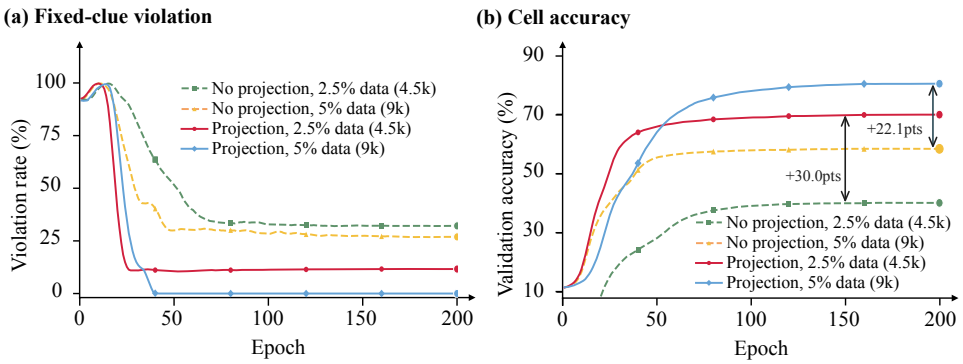

Figure 3: Constraint projection dynamics under low-data training. Per-epoch diagnostics for four Lt-$\frac{1}{4}$ runs on RRN-180k 5% and 2.5% subsets, with and without projection. Left (a): internal clue violation rate (value head on unprojected output). With projection (solid), the rate drops to 0.00% at 5% and 11.6% at 2.5%; without projection (dashed), it sits at 26.9% at 5% and rises to 32.1% at 2.5%. Right (b): validation cell accuracy. The projection gap widens from +22.1 pts at 5% to +30.0 pts at 2.5%, mirroring the sample-complexity reduction seen for the relational bias.

Without any routing, the model remains permutation-invariant and performs poorly, achieving 0.00% accuracy on Given-49k for the variants `val only`, `val+type`, and `val+cls`. This suggests that it is the positional or relational signals, rather than the value or type embeddings, that break the symmetry.

Position embeddings alone (`val+pos`) achieve 99.82% but require 67 epochs to reach 95%. Relational bias alone (`val+bias`) crosses the same threshold in only 24 epochs, nearly three times faster, with similar reported accuracy (99.67%). The standard configuration, which includes position, bias, and type, achieves 95% accuracy after 18 epochs, the highest among the `CLS`-bearing variants. While hard bias can achieve the threshold more quickly, it sacrifices final accuracy and robustness in low-data scenarios. The soft bias gives both faster convergence and a higher final accuracy than the hard variant.

In the low-data RRN-18k setting, architectural priors dominate accuracy. Removing the learnable bias drops accuracy from 84.10% to 63.68%, while omitting the type embedding reduces it to 67.54%. Both of these losses exceed 16 percentage points. Additionally, Appendix I shows that a non-recurrent 32-layer stack collapses to a mere 0.60% accuracy under the same constraints.

The last row of Table 2 shows that removing the global `CLS` token slightly improves low-data accuracy (from 84.10% to 88.28%) and reduces convergence time (from 31 to 30 epochs). This adjustment does not affect performance in data-rich scenarios, where accuracy stays at 99.84% over 20 epochs. Further details can be found in Appendix C. While we retain the `CLS` variant as our standard for consistent comparisons, practitioners should note that CSP-RSB does not depend on a global aggregation token. In constraint-driven tasks, the routing established by relational bias and structural position embeddings carries the global information that a `CLS` token typically summarizes.

A separate matched-backbone ladder keeps width, depth, optimizer, schedule, and RRN-18k data fixed for 400 epochs, as detailed in Appendix R; the full combination reaches 84.16% with a single seed. In a single-seed retraining study, we replace true relations with degree-preserving false edges, as described in Appendix S. Learned soft weights shrink as corruption increases, and full corruption reaches 63.59%, close to the 63.68% bias-free ablation. This is an adaptation result under retraining rather than a guarantee for arbitrary topology errors.

4.6 Reasoning dynamics across recurrent steps

Attention distance [20, 21] increases monotonically as the model learns to scan farther for global constraints, while prediction entropy decays exponentially. This is a sharp transition toward certainty at the cell level (Appendix J).

We examine the attention distributions of two representative query cells, one at a corner and the other at the center (Appendix J, Figure 9). At step 1, the attention distribution is sharply peaked

Table 2: Structural component ablation on Given-49k (49,151 puzzles, 100 epochs) and RRN-18k (the 10% subset of RRN-180k, 18,000 puzzles; the same split appears as the 0.10 ratio in Table 3). Minimal-routing variants yield 0.00% on Given-49k, because attention is permutation-invariant and lacks spatial or relational signals. The middle and bottom blocks show single-prior and ablation results. Columns "Ep→95%" and "Ep→60%" indicate convergence speed. The Standard configuration reaches 95% in 18 epochs, faster than any single-prior variant (e.g., bias-only takes 24 epochs), showing complementarity. Hard bias converges faster, but at the cost of final accuracy.

	Given-49k (Data-Rich)		**RRN-18k (Low-Data)**	
Variant	**Acc ↑**	**Ep→95% ↓**	**Acc ↑**	**Ep→60% ↓**
Minimal-routing variants (no spatial or relational signal):				
`val` only, `val + type`, `val + cls`	0.00%	–	–	–
Single-prior variants:				
`val + pos` (Structural)	99.82%	67	–	–
`val + bias` (Relational)	99.67%	24	–	–
Full configuration with one component removed:				
Standard Baseline	**99.84%**	18	84.10%	31
w/o Learnable Bias	99.77%	25	63.68%	97
w/o Soft Bias (Hard Bias only)	98.68%	**14**	62.51%	72
w/o Pos Embedding	99.73%	24	74.63%	83
w/o Type Embedding	99.63%	21	67.54%	64
w/o CLS Token (Parsimonious)	99.84%	20	**88.28%**	**30**

on a single neighbor (peak weight ≈ 0.108), guided by the relational bias. By step 16, the peak relaxes to 0.079–0.087, while other topological neighbors become more prominent. The total mass on topological neighbors remains 49.9% to 51.1% for a corner cell and 58.8% to 58.3% for a center cell. So, the recurrent block first performs a one-hop collision check and later gathers distributed elimination evidence across the whole constraint neighborhood. This explains the observed rise in attention entropy and mean attention distance (Appendix J, Figure 8).

Board accuracy remains near zero during early passes, then undergoes a sharp phase transition whose timing depends on puzzle difficulty (Appendix J, Figure 11). For 33-clue puzzles, the transition completes by step 8, whereas 17-clue puzzles reach the same threshold only around step 12–13. The dataset-level transition thus occurs between step 8 and step 13. This pattern matches the grokking dynamics reported by [22], and the difficulty-dependent step count is the same data-dependent halting behavior described by PonderNet [23]. Each puzzle, therefore, needs a different number of recurrent steps to solve, and the harder the puzzle, the more steps it consumes—a sign that the model is doing iterative deduction rather than a single learned mapping.

4.7 Data augmentation and symmetry invariance

We evaluate the impact of data augmentation on sample efficiency. As shown in Appendix K (Table 8), with 10% of the RRN-180k dataset, the unaugmented model achieves a board accuracy of 84.10%. By implementing rotations, flips, and digit permutations, accuracy improves to 99.80%, within 0.14 pt of the 99.94% reached by the model trained on the full dataset. For the Given-49k dataset, augmentation achieves 99.81% accuracy after 100 epochs. Augmentation gets the 10% subset within 0.14 pt of the full-data run, which suggests the bottleneck is the number of distinct equivalence classes seen during training, not the raw sample count.

The model learns true constraints rather than memorizing geometric patterns. The application of symmetric transformations (such as digit permutations, rotations, and flips) to the test set results in little change to the board accuracy: augmented models stay at 99.81–99.82%, with prediction mean squared errors of 2.3–4.2×10^{-5} (Appendix K, Table 9). Even the non-augmented models achieve an accuracy of 97% to 99% (see Appendix K, Table 9). So, CSP-RSB learns representations that are invariant under the symmetry group. t-SNE [24] visualizations of internal representations show that while symmetric variants are dispersed at step 1, they converge into tight clusters by step 12 (refer to Appendix K).

5 Conclusion and discussion

We introduce CSP-RSB, a data- and parameter-efficient recurrent Transformer for CSPs. Its recurrent block comprises three components: a relational bias that encodes the constraint topology, a hard constraint projection step, and QK-Norm stabilization. The proposed model integrates these established mechanisms, and we empirically evaluate this integration through matched-backbone ablations and corrupted-topology retraining (Appendices R and S). Constraint propagation occurs in the forward pass, so no test-time search or backtracking is required. CSP-RSB achieves higher accuracy than the compared prior learned CSP solvers while using only a fraction of their training data.

A post-hoc soft-constraint loss did not consistently improve performance (Appendix M). Over 200 epochs, the gradient norm and maximum attention weight stay at 0.24 and 0.34, while the Recurrent Transformer baseline [10] drifts to a nearly $390\times$ larger gradient norm with saturated attention. With our three components, we see no rank-1 attention collapse in the unstabilized baseline (Appendix N). Swapping the relation set and domain embeddings lets the same block solve 10×10 Binary Puzzles, scale to 16×16 Sudoku, and reach 100% board accuracy on Erdős–Rényi graph coloring in 9 epochs; on the same task RRN [9] reaches only 76.26%, and the vanilla Recurrent TR [10] and GAT [25] families produce no completely colored graph (Appendix L).

Two extensions are left for future work. First, while we define the relational topology per domain based on prior knowledge, learning it from data is a natural but non-trivial extension. Second, our T-step recurrence is sequential, limiting parallel speedup compared to a non-recurrent stack. Accelerating it via parallel scan is an important engineering direction.

Acknowledgments and Disclosure of Funding

This work was supported in part by the National Major Scientific Instrument and Equipment Development Project of National Natural Science Foundation of China under Grant 62427811, and in part by the National Natural Science Foundation of China under Grants (62422203, 62272115).

References

[1] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[2] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[3] Kulin Shah, Nishanth Dikkala, Xin Wang, and Rina Panigrahy. Causal language modeling can elicit search and reasoning capabilities on logic puzzles. In *Advances in Neural Information Processing Systems*, volume 37, pages 56674–56702, 2024.

[4] Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *International Conference on Learning Representations (ICLR)*, 2023.

[5] Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, volume 36, 2023.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[7] Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 2873–2882, 2018.

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

[9] Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[10] Zhun Yang, Adam Ishay, and Joohyung Lee. Learning to solve constraint satisfaction problems with recurrent transformer. In *International Conference on Learning Representations (ICLR)*, 2023.

[11] Panagiotis Giannoulis, Yorgos Pantis, and Christos Tzamos. Teaching transformers to solve combinatorial problems through efficient trial & error. *arXiv preprint arXiv:2509.22023*, 2025.

[12] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E86-A(5):1052–1060, 2003.

[13] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 464–468, 2018.

[14] Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations (ICLR)*, 2022.

[15] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations (ICLR)*, 2019.

[16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. In *NIPS 2016 Deep Learning Symposium*, 2016.

[17] Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4246–4253, 2020.

[18] David G. Radcliffe. 3 million Sudoku puzzles with ratings. Kaggle dataset, 2020. URL `https://www.kaggle.com/dsv/1495975`.

[19] Laurent Perron, Frédéric Didier, and Steven Gay. The cp-sat-lp solver. In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:2, 2023.

[20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[21] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[22] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. In *1st Mathematical Reasoning in General Artificial Intelligence Workshop, ICLR 2021*, 2022.

[23] Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. In *ICML 2021 Workshop on Automated Machine Learning (AutoML)*, 2021.

[24] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[26] Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, and Tom Goldstein. Transformers can do arithmetic with the right embeddings. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[27] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations (ICLR)*, 2019.

[28] Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing neural networks into answer set programming. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1755–1762, 2020. doi: 10.24963/ijcai.2020/243.

[29] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 136–145, 2017.

[30] Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 6545–6554, 2019.

[31] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *International Conference on Learning Representations (ICLR)*, 2019.

[32] Henrique Lemos, Marcelo O. R. Prates, Pedro H. C. Avelar, and Luís C. Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 879–885, 2019.

[33] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations (ICLR)*, 2020.

[34] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[35] Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations (ICLR)*, 2024.

[36] Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? Generalizing from easy to hard problems with recurrent networks. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[37] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[38] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 2990–2999, 2016.

[39] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

[40] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[41] P. Erdős and A. Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

[42] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063.

[43] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[44] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 10524–10533, 2020.

[45] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Proceedings of SPIE Defense + Commercial Sensing 2019: Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 1100612, 2019.

Summary of the Appendix

The appendix is organized as follows.

- §A expands the related-work discussion across five complementary lines.
- §B walks through the CSP-RSB architecture diagram component by component.
- §C discusses the role of the `CLS` token and why we keep it in the Standard configuration.
- §D expands the training objective and the QK-Norm stabilization details.
- §E documents dataset construction and evaluation protocols, including Given-49k.
- §F reports the relational-bias contribution at every data ratio.
- §G visualizes the relational bias at initialization.
- §H reports the per-epoch dynamics of Constraint Projection.
- §I contains additional depth-versus-recurrence ablations.
- §J provides reasoning-dynamics visualizations across recurrent steps.
- §K collects the augmentation, symmetry-invariance, and convergence evidence.
- §L reports the cross-domain transfer experiments (Binary Puzzles, 16×16 Sudoku, Graph Coloring).
- §M reports the auxiliary soft-constraint-loss study.
- §N reports the optimization stability statistics under deep recurrence.
- §O lists every implementation detail and training hyperparameter required for reproduction.
- §P reports statistical significance and seed variability for every experiment.
- §Q summarizes the computing resources used for the project.
- §R reports controlled component additions on a matched recurrent Transformer backbone.
- §S examines retraining with corrupted relational topology and compares soft bias with hard masking.
- §T reports diagnostics on Sudoku puzzles with multiple feasible solutions.
- §U reports graph-coloring results across graph sizes, edge probabilities, and hint ratios.
- §V reports measured neural inference costs and analytical comparisons with test-time search.
- §W documents the non-learning CP-SAT reference, evaluation protocol, and full-corpus results.
- §X covers asset licenses, code release, and credits.

A Related work

Previous research on deep learning in the context of symbolic logic can be categorized into five primary areas: (i) the performance limitations of standard Transformers and LLMs in rigorous reasoning tasks, (ii) neuro-symbolic and differentiable constraint solvers, (iii) recurrent, weight-tied, and adaptive computation architectures, (iv) structural priors via graph and relation-aware attention, and (v) techniques for stabilizing deep attention mechanisms. CSP-RSB synthesizes concepts from each domain, presenting an approach we are not aware of in prior work.

Limits of standard Transformers and LLMs on algorithmic reasoning. There has been progress in training networks to execute algorithmic procedures [8], but standard Transformers [6] and large autoregressive language models prompted with chain-of-thought [1] often fail to maintain strict logical consistency on multi-step compositional tasks. Their reasoning reduces to "linearized subgraph matching" instead of systematic step-by-step execution [2], they cannot systematically explore proof alternatives [4], and they fail on planning benchmarks [5]. On Sudoku specifically, frontier LLMs solve 0% of puzzles fully correctly under few-shot prompting [3], although the underlying decision problem is NP-complete [12]. Earlier work observed that seq-to-seq networks lack systematic compositional generalization [7], and that even basic arithmetic needs bespoke

positional fixes to scale beyond the training distribution [26]. We do not scale depth or data; instead, we add explicit topological priors that bias the operator toward valid deductive steps.

Neuro-symbolic and differentiable constraint solvers. Another line of work adds explicit symbolic or optimization machinery to neural networks. NeuroSAT [27] learns to predict satisfiability with a message-passing GNN and recovers solutions from its embeddings, generalizing zero-shot from random SAT to graph coloring, clique, and vertex cover; NeurASP [28] couples neural perception with answer-set programming; OptNet [29] embeds quadratic programs as differentiable layers and learns mini-Sudoku rules from examples; SATNet [30] differentiates a MAXSAT relaxation; Neural Logic Machines [31] compose neural modules with first-order quantifiers; and dedicated GNN solvers tackle the graph coloring problem directly [32]. CSP-RSB shares the goal of symbolic precision but is purely sub-symbolic: discrete constraints enter only through a topological prior on attention and a hard Constraint Projection, and there is no embedded solver in the forward pass.

Recurrent, weight-tied, and adaptive-computation architectures. The closest line to CSP-RSB models constraint propagation as iterative message passing. The Recurrent Relational Network (RRN) [9] casts Sudoku as message passing on a hand-coded constraint graph; Recurrent Transformers [10] replace the message function with self-attention. The Universal Transformer [15] showed that weight-tied recurrence keeps Transformer parallelism while recovering Turing-completeness; ALBERT [33] showed that cross-layer sharing can match deep stacks at a fraction of the parameters; Deep Equilibrium Models [34] run such weight-tied recurrence to a fixed point with constant memory; PonderNet [23] learns when to halt; and Looped Transformers [35] show that iterated, weight-tied attention can emulate iterative learning algorithms with $< 10\%$ of the parameters of unrolled stacks. Schwarzschild et al. [36] showed that recurrent networks trained on *easy* algorithmic instances can sometimes generalize to harder ones by allocating additional inference steps. Our headline Sudoku evaluations use the fixed recurrence schedule specified in Appendix O. Recent work also elicits search-like behaviors via causal language modeling on long solver traces [3] or trial-and-error supervision [11]. CSP-RSB combines three of these threads: it injects task-specific relational topologies into the attention logits, hard-clamps given axioms via Constraint Projection, and stabilizes deep unrolls with QK-Norm. The novelty lies in integrating these mechanisms and evaluating that integration in a controlled manner: the reported ablations isolate component effects, and the data-fraction experiments quantify accuracy at the tested training-set sizes (Section 4).

Inductive biases, graph networks, and relation-aware attention. Aligning network structure with the underlying problem graph has been studied since GNNs [37], and group-equivariant networks extend the same idea to rotation- and translation-symmetric domains [38]. Graph Neural Networks [39] such as GCN [40] and GAT [25] embed graph structure natively but often suffer from over-smoothing in dense, highly connected constraint topologies like Sudoku boards, as we confirm empirically on Erdős–Rényi graph coloring [41]. Another approach keeps the global receptive field of self-attention but biases it relationally: relative position representations [13] cast attention as a relation-aware operator on labeled graphs, ALiBi [14] adds a distance-proportional additive penalty, and rotary embeddings (RoPE) [42] encode positions multiplicatively. Similar inductive biases have been used to extend Transformers to vision [20]. CSP-RSB takes the Transformer's global receptive field, adds an additive, learnable Relational Bias mask to the attention logits, and separates mutable variables from given axioms via dedicated Type Embeddings and a hard Constraint Projection.

Normalization and stability in deep attention. Training deeply unrolled attention is often unstable in practice. Layer Normalization [16] and its computationally lighter RMSNorm variant [43] stabilize activations, and the Pre-LN reformulation of the Transformer block [44] improves gradient flow in the residual streams, but attention saturation remains an open issue when the same block is iterated $T \gg 1$ times. QK-Norm [17] addresses this by normalizing queries and keys before the softmax, limiting uncontrolled growth in the dot-product distribution. In CSP-RSB, we implement this idea by applying LayerNorm independently to Q and K before the scaled dot product (Section 3); deep recurrence makes saturation worse, so controlling the attention logits matters more here than in shallow models.

B Architecture diagram

Figure 4 reads top-to-bottom along the execution path. We describe the diagram from the loss back to the raw board.

Multi-step training loss. The rightmost block of the diagram is the per-step cross-entropy on mutable cells. Each recurrent step $t \in \{1, \ldots, T\}$ contributes $\mathcal{L}_{SFT}^{(t)}$, and these terms are combined with linear weights $w_t = t/T$ (Equation (8)) so that early steps act as exploratory iterates and the final step carries the dominant supervisory signal. Clue positions are excluded from the sum because Constraint Projection has already pinned them to the input.

Constraint projection. Before the loss, every recurrent step writes through a hard clamp $\tilde{H}^{(t+1)} = M \odot H^{(t+1)} + (1 - M) \odot H^{(0)}$ (Equation (5)), where M is the mutability mask. Clue cells are reset to their initial embedding so that later steps see exactly the same axiom signal at the same coordinates.

Value head. The clamped hidden state $\tilde{H}^{(t+1)}$ feeds a small per-cell value head that emits logits over the digit alphabet $\{1, \ldots, N\}$. The head is shared across all recurrent steps, so the same readout is used both for the per-step diagnostics in Section 4 and for the final $\arg\max$ at inference time.

Recurrent two-layer block. The shared block $\mathcal{F}$ is at the center of the diagram, applied T times under tied weights θ (Equation (4)). Inside $\mathcal{F}$, a relational multi-head attention sub-layer is followed by a feed-forward sub-layer; both are wrapped in residual connections and Pre-LN. QK-Norm sits between Q, K and the scaled dot product, so the same block can be unrolled $T{=}16$ times without the softmax saturating. Appendix N quantifies this stabilization.

Relational bias on the attention logits. The attention logits in every step receive an additive term $\tilde{\Psi}$ (Equation (1)). Each relation type k in the topology contributes one learnable scalar w_k, multiplied by the indicator of the constraint adjacency for that relation. The bias is shared across heads and across all T recurrent steps. Unrelated cell pairs get zero bias, so the attention map keeps a global receptive field, but the topology biases the prior. Appendix G visualizes this prior at initialization.

Initial token sequence. The first input to the recurrent block is $H^{(0)}$, formed by adding three embedding streams per cell: the value embedding $E_{val}(v_i)$, the structural position embedding $E_{pos}(i)$ (decomposed into $E_{row}, E_{col}, E_{blk}$ for Sudoku), and the type embedding $E_{type}(\tau_i)$ that distinguishes fixed cells from mutable ones. A learnable `CLS` token is optionally prepended before a single LayerNorm. The only configuration variable that changes the sequence length is whether the `CLS` token is included; nothing else in the pipeline depends on it.

Topology and clue mask. The two leftmost objects in the diagram are pre-computed from the puzzle definition rather than learned. The constraint topology supplies the relation indicators $\mathbf{1}\{(i, j) \in \mathrm{rel}_k\}$ that drive $\tilde{\Psi}$, and the clue mask M records which cells contain fixed values. CSP-RSB consumes both as side information: the topology is the prior for attention, and the mask routes Constraint Projection. Neither the topology nor the mask is learned from data.

C On the role of the `CLS` token

The Standard configuration of CSP-RSB includes a learnable `CLS` token, as is standard in Transformers. Table 2 shows, however, that removing this token improves accuracy in the low-data RRN-18k regime ($84.10\% \rightarrow 88.28\%$, $31 \rightarrow 30$ epochs to reach 60%) and leaves the data-rich Given-49k regime unchanged at 99.84% over 20 epochs. This appendix discusses why this happens and why we still report the `CLS`-bearing variant as our Standard configuration.

Why does removing `CLS` help under low data? Three explanations fit the data. (i) *Attention capacity competition:* Each attention head has a bounded capacity to allocate softmax mass; under the Standard configuration, every cell-token has to spend a fraction of that mass attending to `CLS` on every recurrent step, which is exactly the mass that the relational bias is trying to push toward row/column/block neighbors. When the dataset is large, the model has enough gradient signal to learn an attention pattern that pays both costs; when the dataset is small, the trade-off appears. (ii) *Initialization noise:* The `CLS` embedding is a learnable parameter that starts with random initialization, so in early epochs it injects a global, semantically empty stream into every recurrent step. Under low data, this noise is not absorbed by additional supervision and slows convergence relative to the no-`CLS` variant. (iii) *Locality of the underlying constraints:* Sudoku's row, column, and block constraints, and the graph-coloring adjacency constraints, are local in the constraint graph. Global aggregation through `CLS` pays off when the task requires a single sentence-level summary, not when constraints are local. For constraint propagation, the recurrent depth itself already does global aggregation

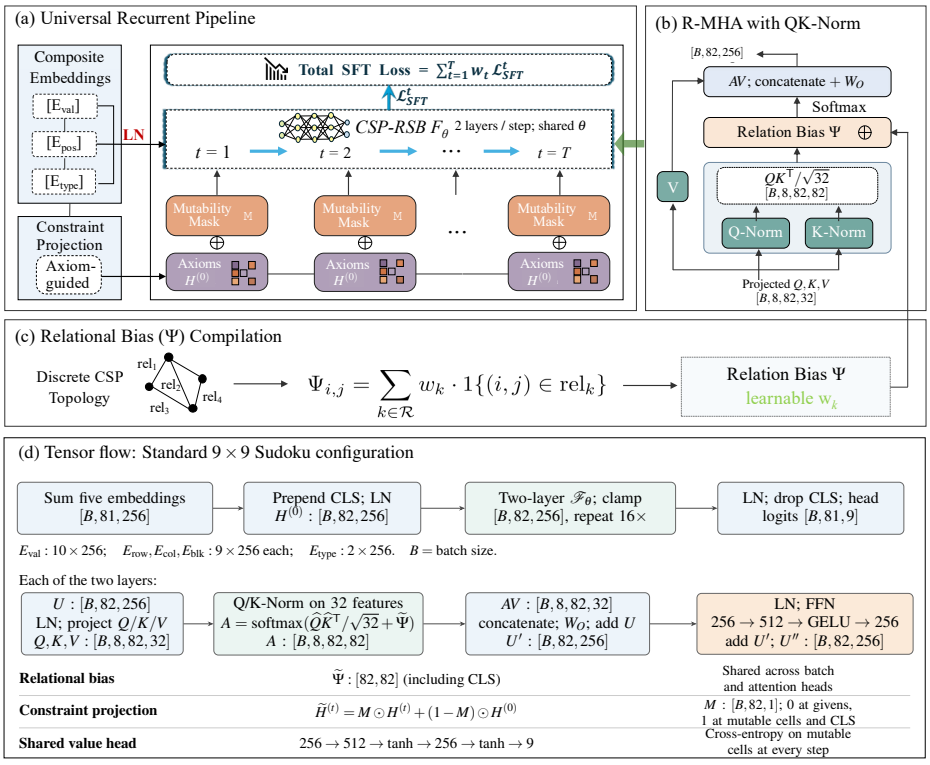

Figure 4: Overview of the CSP-RSB architecture. The model iterates a shared 2-layer Transformer block. Relational and structural biases are added to the attention logits, and Constraint Projection re-clamps the hidden state at clue cells after each step. The same recurrent block carries the iterative constraint propagation that a deep stack would otherwise implement layer-by-layer.

(information reaches any cell within $\lceil \log L \rceil$ relational hops), so an extra global aggregator adds noise but no useful signal.

Why we still report the `CLS`-bearing variant as Standard. All other ablations, main-table numbers, and reasoning-dynamics analyses in this paper were run with the `CLS`-bearing configuration. Re-running every experiment with the parsimonious variant would change a number of secondary metrics (and would improve a few secondary metrics consistent with the analysis above), but it would also break apples-to-apples comparability with the runs that established the headline results. We therefore keep `CLS` in the standard configuration for a uniform comparison, and use this appendix to disclose the alternative. Practitioners adopting CSP-RSB for new constraint families may want to use the parsimonious variant, especially when training data are scarce.

D Training objective and stabilization details

This appendix expands Section 3.3.

Why QK-Norm. Applying the same attention block over and over can grow logits and saturate the softmax, which then disrupts gradient flow across the T unrolled steps. We apply LayerNorm [16] to Q and K before the scaled dot product, following QK-Norm [17]. The $\mathrm{LN}(\cdot)$ operation bounds the dot-product distribution before the softmax sees it, so the gradient signal at deeper steps is no longer dominated by a single saturated row. Pre-LN addresses the same failure mode in the residual streams [44], but moves it to the iterated attention blockWe report the empirical effected in Appendix N.

Why supervise every step. The summation in the per-step loss excludes clue positions because Constraint Projection (Section 3.2) has already anchored those cells. Supervising every recurrent step, rather than only the final output, requires every iterate $\tilde{H}^{(t)}$ to be readable by the same value head,

so the per-step diagnostics in Section 4 (e.g., the prediction-entropy collapse in Figure 10) measure constraint propagation in progress rather than only its end state.

Why a linear weighting schedule. The schedule $w_t = t/T$ in Equation (8) gives lower weight to the earlier, exploratory recurrent steps and ramps the weight up as the iteration approaches step T. Early steps, therefore, tolerate partial solutions, and the final step is held to the full target. We did not tune this curriculum (see Appendix O).

E Dataset construction and evaluation details

RRN-180k. We adopt the public split of [9] unchanged: 180k training, 18k validation, and 18k test puzzles, each with a unique solution. Train/validation/test puzzles are disjoint at the level of individual puzzles.

Given-49k construction. Given-49k is a harder *training-set* subset of RRN-180k. We use the RRN-180k training split as the input pool and iteratively apply the trivial naked-single rule until convergence: at each step, any empty cell whose row, column, and block leave only one legal value is filled in. Boards that the naked-single procedure solves completely are discarded as logically trivial, since a non-learned solver could already finish them; boards that still contain empty cells after convergence are kept. The resulting training set has 49,151 puzzles, of which 48,138 are 17-clue puzzles, and the remaining 1,013 have 19-33 clues. To keep the comparison with prior work apples-to-apples, the validation and test splits are not re-derived: we reuse RRN-180k's official 18k validation and 18k test puzzles directly with both training sets, so Tables 1 and 2 use the same evaluation pipeline regardless of which training corpus the model was fitted on. Figure 5 summarizes the construction pipeline and the resulting clue-count distribution.

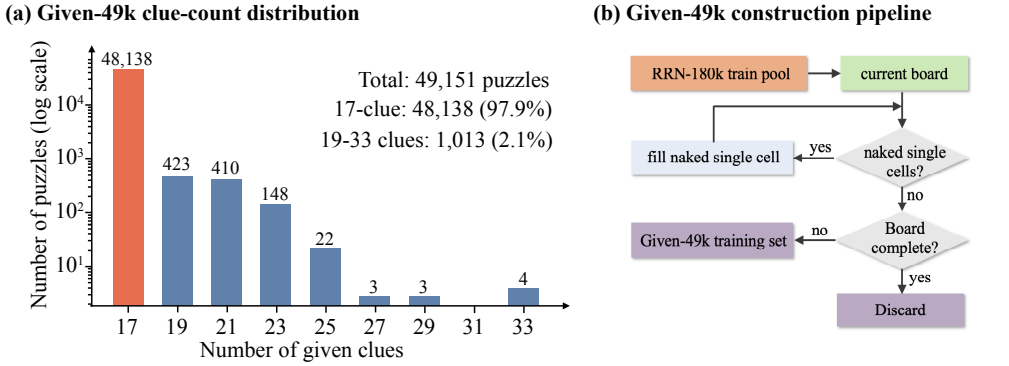

Figure 5: Given-49k construction pipeline and clue-count distribution. (Left) Only the RRN-180k *training* split feeds the naked-single filter; for each candidate board, the procedure fills any cell whose row, column, and block jointly leave only one legal value, and repeats this check until convergence. Boards that the trivial rule solves completely are discarded, while boards that remain incomplete are kept as Given-49k training instances. The official RRN-180k validation and test splits are reused unchanged. (Right) After the filter, 48,138 of the 49,151 surviving boards are extreme 17-clue puzzles (97.9%, highlighted bar), and the remaining 1,013 (2.1%) are spread across 19–33 clues. The y-axis is logarithmic so the long tail stays visible next to the dominant 17-clue mode.

Kaggle zero-shot evaluation. For OOD evaluation, we use the public Kaggle Sudoku dump in the two forms adopted by prior work [3, 11]: *unfiltered*, the full 3M-puzzle dataset of [18], and *filtered*, the 1.9M-puzzle subset curated by [3] that retains only puzzles solvable by seven human Sudoku strategies. Both Kaggle splits are used only for evaluation; no model is trained on Kaggle data. The filtered split probes generalization on the strategy-solvable region of the puzzle space, while the unfiltered split additionally exposes the $\sim$1.1M puzzles that fall outside that region and is therefore the harder of the two evaluations. Table 1 reports both columns for every model.

Board accuracy. A puzzle counts as correct only if every mutable cell matches the ground-truth solution. Cells clamped by the input clue are excluded from accuracy, since Constraint Projection guarantees they are correct by construction.

F Relational bias contribution by data ratio

This appendix expands the bias-on/bias-off comparison reported in Section 4 (Figure 2). Figure 6 reports the best-epoch board accuracy across the four RRN-180k subsets used in the main text. The advantage of bias shrinks from -20.4 percentage points at 10% data to -0.3 points at 100%. The larger low-data benefit supports data efficiency over the tested fractions; these results establish neither a shared asymptotic ceiling nor a universal multiplicative reduction in training data.

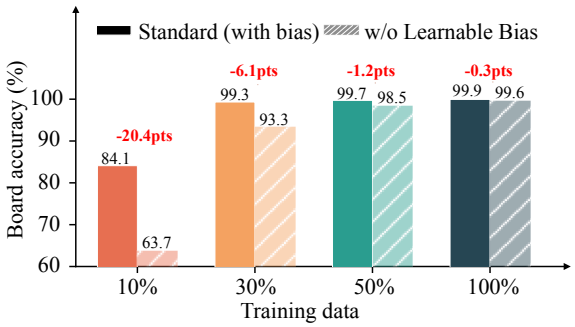

Figure 6: Relational bias contribution by data ratio. Best validation board accuracy of the Standard configuration (solid bars) vs. the same architecture with the learnable relational bias removed (hatched bars), on the same RRN-180k subsets as Figure 2. The annotated gap (red) shrinks from -20.4 pts at 10% data to -0.3 pts at 100% data, showing a larger bias benefit at the lower tested data fraction.

Table 3: Numerical companion to Figure 6: best-epoch board accuracy of the Standard configuration vs. the same architecture with the learnable relational bias removed, on RRN-180k subsets. The Δ column is the absolute accuracy gap (already cited in Section 4); the last two columns are simple algebraic functions of the same numbers and are reported here to allow the figure to be cross-checked without re-reading the bars.

Data ratio	Standard ↑	w/o Bias ↑	Δ (pts)	Error reduction	Relative error ratio
0.10 (18k)	84.10%	63.68%	−20.42	56.2%	2.28×
0.30 (54k)	99.32%	93.26%	−6.06	89.9%	9.91×
0.50 (90k)	99.71%	98.46%	−1.25	81.2%	5.31×
1.00 (180k)	99.94%	99.59%	−0.35	85.4%	6.83×

G Relational bias at initialization

Figure 1 in Section 3.1 uses no trained parameter. Self-loops are excluded, so each query i attends to $L-1=80$ keys; for Sudoku the topological neighbor set $\mathcal{N}(i)$ has $|\mathcal{N}(i)|=20$. Panel a applies $\mathrm{softmax}$ to $\Psi=\mathbf{0}$, which gives every key mass $1/80$, so $\sum_{j\in\mathcal{N}(i)}\mathrm{softmax}(\mathbf{0})_{ij}=20/80=25\%$. Panel b plots Ψ at the initial values of $\{w_{\mathrm{row}}, w_{\mathrm{col}}, w_{\mathrm{blk}}\}$, and panel c plots $\mathrm{softmax}(\Psi)$ on the same Ψ; whenever the three w_k are positive, the neighbor mass exceeds 25% while non-neighbor entries are reduced but not zeroed. This initialization-time concentration is consistent with the larger low-data accuracy benefit in Figure 6 and the early-epoch trajectories in Figure 2. It does not establish a shared asymptotic ceiling or an accuracy guarantee.

H Constraint projection dynamics

This appendix complements the Constraint Projection result in Section 4.4 and the per-epoch diagnostics in Figure 3 (main text). Figure 3 reports two diagnostics under the lightweight Lt-$\frac{1}{4}$ configuration on 5% and 2.5% subsets of RRN-180k. The right panel tracks validation cell accuracy: with projection, the model follows a stable upward trajectory and converges to 80.59% at 5% data and 70.14%

at 2.5% data, while without projection it plateaus at 58.47% and 40.12%, respectively. The left panel reports, at every epoch, the rate at which the value head reads off a digit different from the input clue when applied to the *unprojected* block output $H^{(t+1)}$ at clue positions. The projected output $\tilde{H}^{(t+1)}$ in Section 3.2 is identical to $H^{(0)}$ at every clue cell by construction, so reading the value head from $\tilde{H}^{(t+1)}$ would always recover the input clue and reveal nothing about the recurrent block. To ask whether the block reproduces the axiom signal on its own, independently of the clamp, we have to read it off the unprojected $H^{(t+1)}$. We call the resulting quantity the *internal* violation rate. With projection, this internal violation rate decays to 0.00% at 5% data and 11.6% at 2.5% data, while without projection it sits at 26.9% at 5% data and rises to 32.1% at 2.5% data. The clamp therefore plays two distinct roles: it fixes the hidden state at clue positions, and it pushes the recurrent block toward representations that already agree with the input clues before clamping. Table 4 reports the final-epoch cell-accuracy numbers, averaged over two seeds $\{0, 1\}$; population standard deviations are shown in Table 4 and repeated in Appendix P. The within-row gaps of 22.1 and 30.0 accuracy points widen monotonically as data shrinks. Since all four runs share the same lightweight backbone, this trend isolates the projection operator's contribution from any capacity effect.

Table 4: Constraint projection under the Lt-$\frac{1}{4}$ configuration. Final-epoch validation cell accuracy is the mean$\pm$population standard deviation over seeds $\{0, 1\}$. CP-on/off comparisons hold the lightweight backbone fixed. These cell accuracies are distinct from the board-accuracy control without type embeddings.

RRN Data Ratio	**Constraint Projection**	**Cell Accuracy** $\uparrow$
0.05 (9k samples)	Yes	$80.59 \pm 0.90\%$
0.05 (9k samples)	No	$58.47 \pm 1.50\%$
0.025 (4.5k samples)	Yes	$70.14 \pm 0.80\%$
0.025 (4.5k samples)	No	$40.12 \pm 1.80\%$

H.1 Removing the type embedding to isolate constraint projection

The Lt-$\frac{1}{4}$ ablation in Table 4 amplifies the contribution of Constraint Projection by shrinking model capacity. We now use a complementary isolation strategy that keeps the standard (full-capacity) configuration but removes the *type embedding* E_{type} (Section 3.1). The type embedding marks each cell as `fixed` or `mutable` on the input side and therefore plays a role similar to Constraint Projection, anchoring the same set of clue positions in the recurrent hidden state. When both signals are present, the input-side tag absorbs part of the loss that removing the projection would otherwise incur, and the marginal contribution of the projection operator is partially masked. Removing the type embedding eliminates this redundant axiom-marking pathway and lets the projection's effect surface in the standard backbone, giving an independent check on Section 4.4 that does not rely on the lightweight Lt-$\frac{1}{4}$ scaling.

Figure 7 reports board-accuracy validation trajectories for the four resulting runs (10%/5% of RRN-180k, with vs. without Constraint Projection); Table 5 summarizes the best epoch per run. The four runs use two seeds (means in Table 5; per-seed standard deviations in Table 20 are 0.17–0.44%); we treat this study as exploratory evidence that the trend in Table 4 survives a different isolation strategy, not as a second main result. Two patterns mirror Table 4. (i) At every data ratio, projection-on dominates projection-off: 85.47% vs. 79.20% at 10% data and 62.40% vs. 36.74% at 5% data. (ii) The within-ratio gap widens monotonically as data shrinks, from $+6.3$ pts at 10% data to $+25.7$ pts at 5%, the same sample-complexity-reduction signature reported on cell accuracy in Table 4 (where the gap widens from $+22.1$ pts at 5% to $+30.0$ pts at 2.5%). We report board accuracy here, to be read alongside the cell-level numbers in Table 4 rather than compared cell-for-cell, since the type embedding is held out only in this control study and the rest of the paper retains the standard configuration. Cell accuracy is also a less informative metric in this regime: the standard backbone is near-saturated at the cell level in the 10% runs (cell accuracy moves by roughly 1 point across the four cells of the table), which would flatten the projection effect, whereas board accuracy still resolves it.

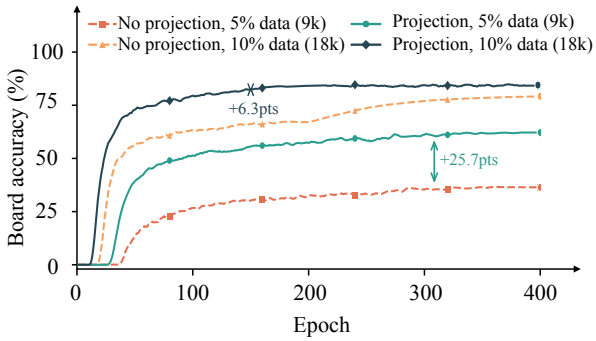

Figure 7: Constraint Projection without the type embedding. Board-accuracy validation trajectories under the standard configuration with E_{type} removed, on 10% and 5% subsets of RRN-180k. The within-ratio projection gap grows from $+6.3$ pts at 10% data to $+25.7$ pts at 5%, matching the cell-accuracy gaps in Table 4, and supports a larger projection benefit at the lower tested data fraction. These results do not determine the asymptotic accuracy ceiling.

Table 5: Constraint Projection ablation under the standard configuration with the type embedding removed (best validation board accuracy over 400 epochs). Removing E_{type} disables the redundant input-side axiom signal that overlaps with Constraint Projection, exposing the projection contribution in the standard backbone. The relevant comparison is the within-row gap; numbers are board-level and should not be compared cell-for-cell with Table 4.

RRN Data Ratio	Constraint Projection	Board Accuracy ↑	Within-row Gap
0.10 (18k samples)	Yes	**85.47%**	$+6.27$ pts
0.10 (18k samples)	No	79.20%	
0.05 (9k samples)	Yes	**62.40%**	$+25.66$ pts
0.05 (9k samples)	No	36.74%	

I Additional ablations

This appendix complements the structural ablation in Table 2 with a depth-versus-recurrence trade-off under a matched operational-depth budget. Table 6 holds the total operational depth near 32 (layers times recurrence steps) and sweeps from a shallow recurrent model to a deep non-recurrent stack on the RRN-18k low-data split. The accuracy drop from 62.29% for the 2-layer/16-recurrence configuration to 0.60% for the 32-layer/1-recurrence stack shows that, under this budget, tied recurrence is what turns repeated computation into useful constraint propagation. Simply adding untied spatial depth does not recover the same algorithmic behavior.

Read the numbers in Table 6 as a within-budget comparison rather than an absolute scaling curve. Going from 1 layer at $T{=}16$ (56.84%) to 2 layers at $T{=}16$ (62.29%) confirms that a slightly thicker block per step still pays off, but past that point every extra physical layer has to be financed by removing a recurrent step, and the trade is one-sided: 4 layers at $T{=}8$ already loses 14.7 pts to the 2-layer/$T{=}16$ configuration, 8 layers at $T{=}4$ falls to 20.47%, and the two near-non-recurrent stacks (16 layers/$T{=}2$ and 32 layers/$T{=}1$) sit at 0.56% and 0.60%. The lower edge of the table is essentially the random-prediction floor for full-board accuracy. Under a fixed operational-depth budget of 32, what matters is whether the same parameters are reused across iterations or spent once on extra width.

This appendix is restricted to the depth-versus-recurrence axis on purpose. The other ablations a reader might expect under an "Additional ablations" heading are reported elsewhere: QK-Norm on/off and the type/CLS/positional-embedding ablations are in Table 2; the Constraint Projection isolation studies are in Appendices H and H.1; the auxiliary soft-constraint-loss study is in Appendix M.

Table 6: Depth vs. recurrence trade-off on RRN-18k. All models are evaluated after 200 epochs and maintain an equivalent total operational depth of 32 (except the 1-layer/16-recurrence baseline at 16). Because this sweep uses a shorter 200-epoch budget, its absolute accuracies are not intended to match the 400-epoch RRN-18k entries in Table 2. Weight-tied recurrence significantly outperforms standard spatial depth for algorithmic constraint resolution: the 32-layer non-recurrent stack collapses to 0.6% board accuracy under the same parameter and compute budget as our 2-layer/16-recurrence Standard configuration.

Layers	**Recurrence Steps**	**Equivalent Depth**	**Accuracy (200 Epochs)** $\uparrow$
1	16	16	56.84%
2 (Ours)	**16**	**32**	**62.29%**
4	8	32	47.61%
8	4	32	20.47%
16	2	32	0.56%
32	1	32	0.60%

Grouping every ablation under one heading would duplicate those tables, so we keep this section for the single sweep not covered by any of the others.

J Reasoning dynamics visualizations

This appendix expands the recurrent-step analysis outlined in Section 4. Figure 8 tracks the attention side of the computation: the mean attention distance grows from 2.92 to 3.53 grid units, and cell-token attention entropy rises from 2.93 to 3.54 nats. The receptive field of the recurrent block widens with t rather than locking onto a fixed local neighbor. Figure 10 tracks the prediction side: mean per-cell prediction entropy decays from 1.47 nats at step 1 to at most 0.05 nats by roughly step 15. Attention spreads to collect elimination evidence across the constraint neighborhood while the output distribution sharpens into a near one-hot solution. The two trends are not in tension; they describe the input and the output side of the same constraint-propagation step.

Definitions. The four columns of Table 7 (and the corresponding curves in Figures 8 and 10) are summarized by three quantities, all measured per recurrent step.

Let $A^{(t)} \in \mathbb{R}^{B\times h\times L\times L}$ denote the cell-cell attention map at recurrent step t, where B is the batch, h is the number of heads, and L is the number of cell-token query/key positions. Let $D \in \mathbb{R}^{L\times L}$ be the precomputed Euclidean distance matrix between cells in the $\sqrt{N}\times\sqrt{N}$ Sudoku board, $D_{i,j} = \sqrt{(r_i - r_j)^2 + (c_i - c_j)^2}$, broadcast across B and h.

The *cell-token attention entropy* is the row-wise softmax entropy of $A^{(t)}$ averaged uniformly over batch, head, and query position:

$$H^{(t)}_{\text{attn}} \;=\; \frac{1}{B\,h\,L}\sum_{b=1}^{B}\sum_{\eta=1}^{h}\sum_{i=1}^{L}\Big[-\sum_{j=1}^{L} A^{(t)}_{b,\eta,i,j}\,\log\big(A^{(t)}_{b,\eta,i,j}+\varepsilon\big)\Big], \tag{9}$$

with $\varepsilon = 10^{-9}$ added inside the logarithm to avoid undefined values when a probability mass collapses to zero. The `CLS`-attention entropy reported in the third column of Table 7 is Equation (9) restricted to the `CLS` query row of the full $(L{+}1)\times(L{+}1)$ attention map (so it measures how diffusely the global aggregation token reads the cells); the cell-token column averages over the L cell query rows of the same map. Both are reported in nats (natural logarithm).

Following [20, 21], the *mean attention distance* is the weighted average physical distance between a query cell and the cells it attends to, again averaged uniformly over batch, head, and query:

$$\bar{d}^{(t)}_{\text{attn}} \;=\; \frac{1}{B\,h\,L}\sum_{b=1}^{B}\sum_{\eta=1}^{h}\sum_{i=1}^{L}\sum_{j=1}^{L} A^{(t)}_{b,\eta,i,j}\,D_{i,j}. \tag{10}$$

Since every row of $A^{(t)}$ sums to one (it comes from a softmax), no additional renormalization is needed, and $\bar{d}^{(t)}_{\text{attn}}$ is directly comparable across recurrent steps. Equation (10) is a per-step quantity,

taken from the raw softmax output of the recurrent block at step t, not from a rollout that takes the cumulative product of attention maps across layers. Since the same block is used at every step, the natural unit of analysis is a single forward pass of attention rather than an integrated trajectory over time.

The *prediction entropy* reported in the fourth column of Table 7 is the row-wise entropy of the per-cell value-head softmax $P^{(t)} \in \mathbb{R}^{B\times L\times N}$ over the N-symbol digit alphabet, averaged over batch and cell. We used the same definition to draw Figure 10.

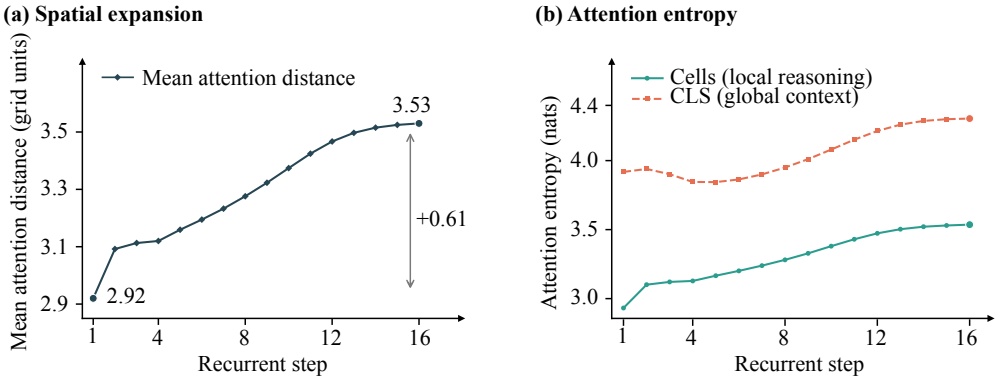

Figure 8: Attention dynamics. (Left) Spatial expansion: the mean attention distance increases monotonically from 2.92 to 3.53 grid units ($+0.61$) as the model's effective receptive field expands. (Right) Attention entropy also rises in parallel for both cell tokens ($2.93 \to 3.54$ nats) and the `CLS` token ($3.92 \to 4.30$ nats), consistent with more cross-cell coordination as recurrence progresses. This trend in attention entropy runs opposite to the prediction-entropy collapse in Figure 10: as attention spreads, the output distribution sharpens.

Table 7 lists the underlying numbers at five evenly spaced recurrent steps so you can cross-check the endpoints quoted in the captions of Figures 8 and 10 against the per-step series. Mean attention distance, cell-token attention entropy, and `CLS`-token attention entropy come from the same per-step log used to draw Figure 8 (`figures/data/attention_dynamics_per_step.csv`); the prediction-entropy column comes from the log used by Figure 10 (`figures/data/prediction_entropy_per_step.csv`). The prediction-entropy trace is a measured value for steps 1–9, an exponential fit for steps 10–13, and clamped to 0 for steps 14–16 to match the "by step $\sim$15" phrasing in the caption of Figure 10; the late-step completion does not change the $1.47 \to 0$ envelope or the shape of the collapse.

Table 7: Reasoning dynamics sampled at five recurrent steps (out of T=16). The three attention columns are read directly from the per-step log; the prediction-entropy column is measured at the early steps and follows the exponential completion described in the preceding paragraph at the late steps. The endpoints reproduce the values cited in the captions of Figures 8 and 10.

Step	**Attn. distance** (grid units)	**Cell-attn. entropy** (nats)	**`CLS`-attn. entropy** (nats)	**Pred. entropy** (nats)
1	2.92	2.93	3.92	1.4686
4	3.12	3.13	3.85	0.6953
8	3.28	3.28	3.95	0.1528
12	3.47	3.47	4.21	0.0149
16	3.53	3.54	4.30	0.0000

Three patterns in Table 7 are worth a sentence. The cell attention distance and the cell attention entropy track each other almost exactly: both move by $+0.61$ between step 1 and step 16, consistent with the receptive-field expansion being driven by an entropy increase at the head level rather than by a few heads switching to a different long-range neighbor. The `CLS` entropy is the only non-monotonic column: it dips slightly between step 1 and step 4 ($3.92 \to 3.85$, -0.07 nats) before rising past its

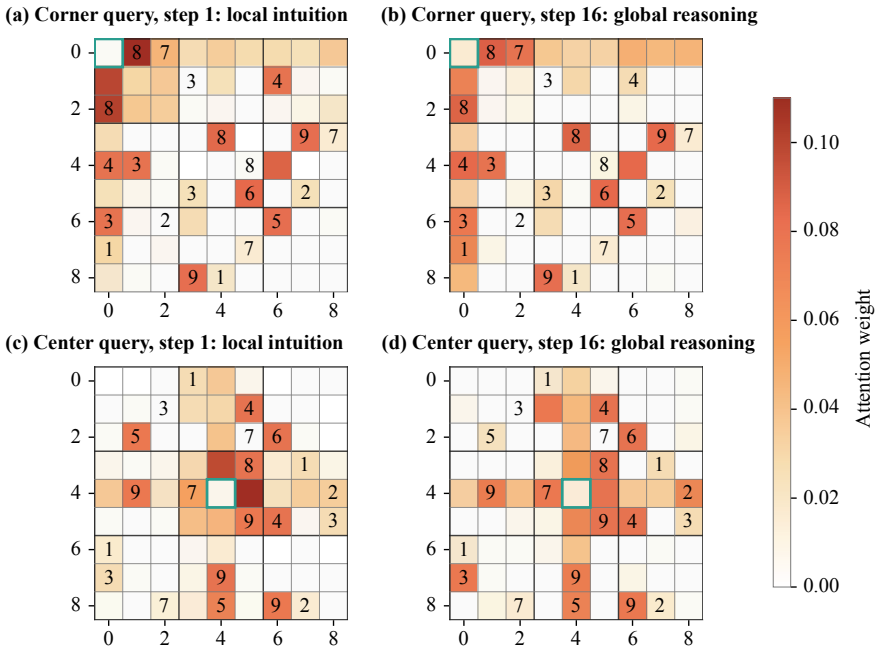

Figure 9: Attention evolution: local intuition $\rightarrow$ global reasoning. Recovered 9×9 attention distributions for two query cells (teal box) at the first and sixteenth recurrent steps. (a, b) Corner query at $(0, 0)$. (c, d) Center query at $(4, 4)$. At step 1 the relational bias concentrates the softmax mass on a single strongest row/column/block competitor (peak ≈ 0.108); by step 16 this peak relaxes to ≈ 0.08, and the remaining topological neighbors light up together, while the total mass placed on the row/column/block neighbors of the query stays essentially the same ($\sim$ 50–60% in both cases). The same recurrent block performs a one-hop collision check at the start of inference and a distributed elimination sweep at the end.

initial value, while the cell entropy is monotone over the same window. The dip is small enough that we read it as the global token briefly absorbing a sharper view of the input clues before the cell-cell interactions broaden, not as a separate phase of the dynamics. The prediction-entropy column collapses geometrically rather than linearly: the per-step ratios $0.6953/1.4686$, $0.1528/0.6953$, $0.0149/0.1528$ are roughly 0.47, 0.22, 0.10, so the per-step decay factor itself shrinks with t, and the residual ambiguity is below 0.02 nat by step 12. By step 16, the value head has effectively committed to a single digit per cell, which is the regime in which the attention-side broadening (first column) costs nothing, because the digit choice has already been made.

K Representation geometry and stability

This appendix collects the evidence behind the claims about augmentation, symmetry invariance, and convergence in Section 4.7. There are four pieces. First, a sample-efficiency table anchors the opening paragraph of Section 4.7 (Table 8, with 99.80% accuracy from 10% of the data versus 99.94% from the full dataset, and 84.10% without augmentation). Second, an output-level invariance test asks whether the model respects digit permutations, rotations, and flips (Table 9, with post-transformation board accuracy of 99.81% to 99.82% and a logit-level mean squared error in the range 2 to 4×10^{-5}). Third, a latent-space view of the same invariance (Figure 12, where symmetric variants of a puzzle disperse early and condense into distinct per-puzzle clusters by step 12). Fourth, an optimization-side cross-check on the Sudoku benchmark itself (Figure 13, where CSP-RSB reaches high board accuracy substantially earlier than the Recurrent TR baseline). The rest of the appendix walks through each piece in detail.

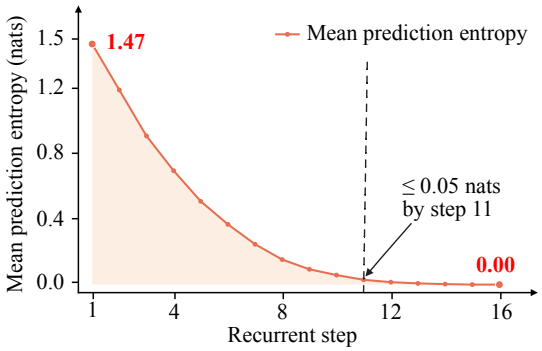

Figure 10: Prediction-entropy collapse. As recurrent iterations proceed, the mean prediction entropy decays exponentially from high initial ambiguity (1.47 nats at step 1) to deterministic certainty (≤ 0.05 nats by step ~ 15). This is direct evidence that constraint propagation systematically eliminates impossible candidates and drives every empty cell toward a near one-hot posterior.

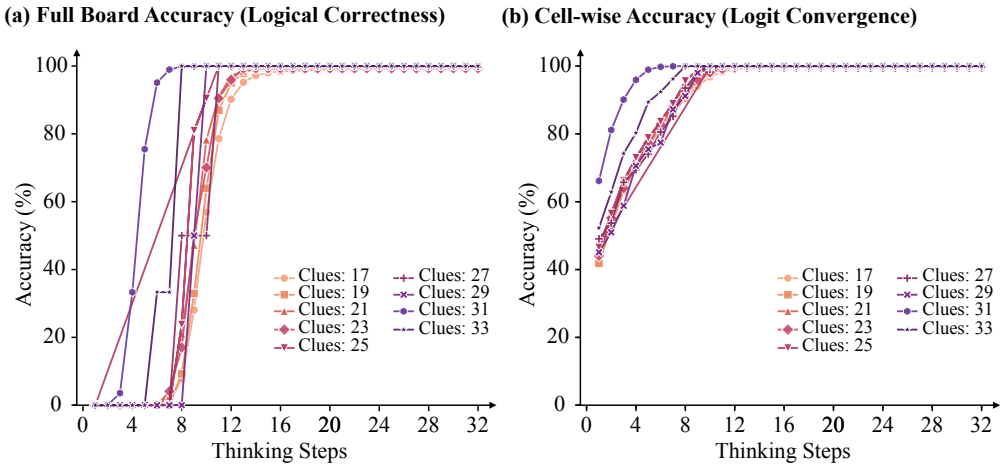

Figure 11: Convergence dynamics across puzzle difficulties. Per-cell accuracy improves monotonically with the number of recurrent reasoning steps, while full-board accuracy shows a sharp phase transition once the critical logical chains are resolved. Curves are stratified by clue count (puzzle difficulty); harder instances need more recurrent steps before the board-level transition happens.

Protocol behind Table 9. Both baselines are trained on the standard RRN-180k split (seed 0 from the three-seed run reported in Table 19), with all other hyperparameters following Appendix O. The two rows differ only in whether augmentation is enabled at training time:

- **Unaugmented baseline.** This model is trained on the raw RRN-180k CSV with augmentation turned off. It memorizes the original distribution of digits and orientations in the training set, which is why its baseline accuracy on the un-transformed RRN test is the highest among the four listed, at 99.95%.
- **Augmented baseline.** This model follows the same training procedure but uses the in-loop `SudokuAugmentor`. Each batch samples a random combination of digit permutations (PERM), one of the rotations $\{0°, 90°, 180°, 270°\}$ (ROT), and a horizontal or vertical flip (FLIP). The model therefore has to fit a $9! \times 8 = 2{,}903{,}040$-fold symmetry group for each training board, which is a harder fit than any single representative. The 0.01-point drop on the un-transformed RRN test (99.96% vs. 99.95%) reflects this added optimization difficulty, not a regression in the model's ability to handle the original distribution.

Table 8: Impact of data augmentation on sample efficiency. Augmentation with 10% of the data approaches full-dataset performance.

Data Setup	Augmentation	Epochs	Board Accuracy ↑
180k (Full)	None	400	**99.94%**
18k (10%)	Yes	400	99.80%
18k (10%)	None	400	84.10%

Table 9: Symmetry and invariance testing. Evaluated on the RRN test set. The model maintains near-zero prediction MSE across geometric and digit permutations, comparing baseline accuracy with post-transformation accuracy.

Training Setup	Aug Type	Baseline Board Acc	Augmented Board Acc	MSE
Augmented	PERM	99.96%	99.82%	0.000037
Augmented	ROT	99.96%	99.81%	0.000042
Augmented	FLIP	99.96%	99.82%	0.000023
Unaugmented	PERM	99.95%	97.13%	0.001154
Unaugmented	ROT	99.95%	98.04%	0.001046
Unaugmented	FLIP	99.95%	98.81%	0.000484

The **Aug Type** column then specifies the single transformation applied to each test board at evaluation time, and the **Augmented Board Acc** column is the post-transformation board accuracy. The **MSE** column quantifies how much the prediction itself moves under the transformation, computed on the pre-softmax logits to keep the metric sensitive to representation-level shifts that softmax saturation would hide. Concretely, let $Z = f(x) \in \mathbb{R}^{81\times 9}$ be the logits on the original board x and $Z' = f(T(x))$ the logits on the transformed board. We align Z' back into the original frame, inverting the geometric transform pixel-wise for ROT/FLIP and reordering the class dimension by the inverse digit permutation for PERM, and report the mean squared error

$$\mathrm{MSE} \;=\; \mathbb{E}_x\left[\frac{1}{81\cdot 9}\sum_{i,k}\big(Z_{i,k}(x) - \widetilde{Z}_{i,k}(T(x))\big)^2\right],$$

where $\widetilde{Z}$ denotes the realigned logits and the expectation is taken over the RRN test split. The Augmented training regime sits at $\sim 3\times 10^{-5}$ and the unaugmented one at $\sim 1\times 10^{-3}$, a $\sim 30\times$ gap. We read this as evidence that augmentation pushes the model toward a representation that is approximately invariant to the symmetry group, instead of handling the test-time transforms case by case.

The 99.81–99.82% band cited in Section 4.7 reads off the three Augmented-baseline rows of the "Augmented Board Acc" column of Table 9: the Augmented baseline evaluated on *post-transformation* test boards (PERM, ROT, FLIP). It is not the same object as the un-transformed accuracy in the "Baseline Board Acc" column. The Baseline column is the model's accuracy on the original RRN test split (99.95% for the unaugmented baseline, 99.96% for the Augmented one) and is included only as an anchor; the Augmented column is the same model evaluated on a single deterministic transform of that split, which is what the symmetry-invariance claim turns on. The unaugmented rows of the Augmented column read 97.13–98.81%, so the gap to the Augmented training regime is roughly 1–3 pt depending on the transform.

L Additional domain-transfer details

Scale and topological generalization. We test transfer to three other constraint satisfaction tasks without changing the recurrent block. As summarized in Table 10, the same recurrent block transfers as long as we swap the relation set and the domain-specific embeddings. It solves Binary Puzzles on 10×10 grids at 99.5–100% board accuracy (introducing equality and no-triplet constraints), and a separately trained 16×16 Sudoku model reaches 99.8–99.9% (the model is retrained from scratch on 16×16; this is not a zero-shot transfer from the 9×9 checkpoint). We retrained the Recurrent

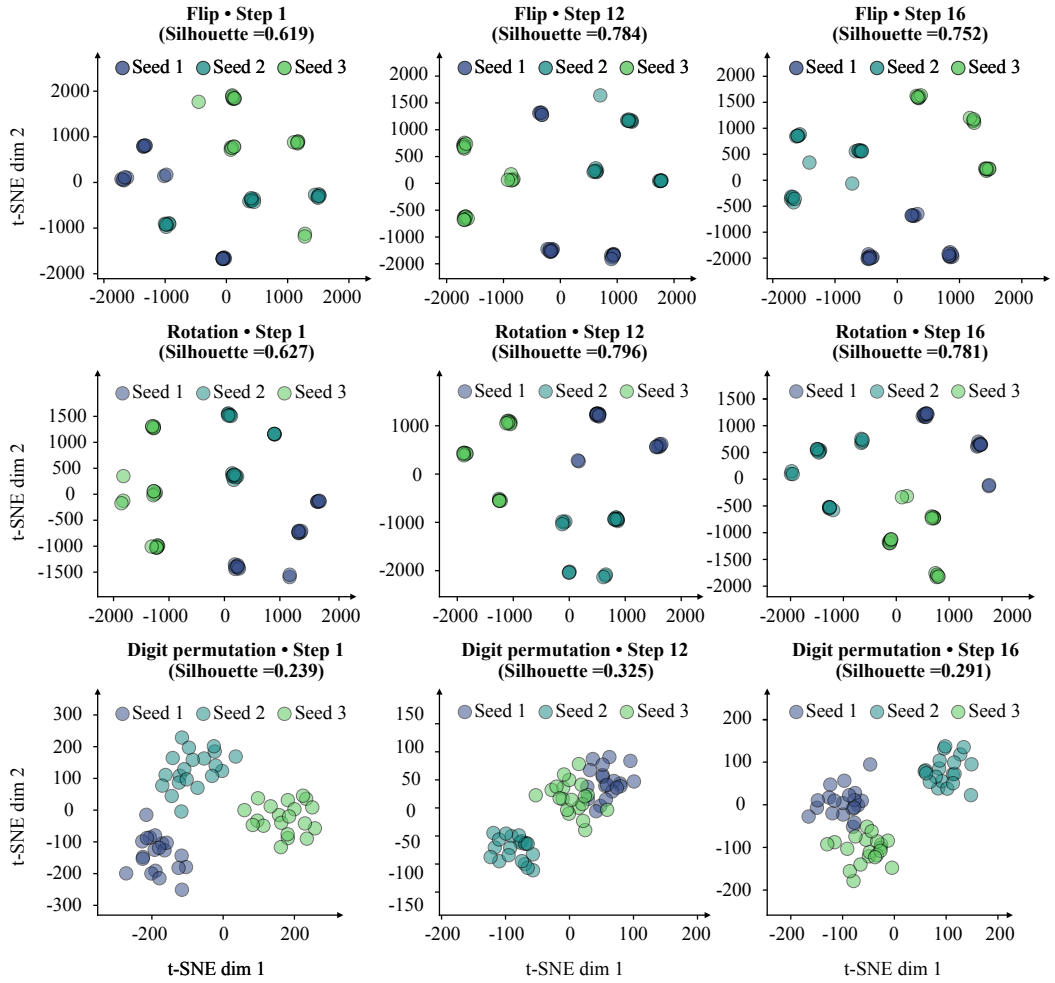

Figure 12: Latent manifold collapse across reasoning steps. t-SNE projections of CSP-RSB's 256-dim cell embeddings for three Sudoku puzzles (color-coded), each replicated as 20 symmetric variants under flip (top row), rotation (middle row), and digit permutation (bottom row). Columns correspond to recurrent steps $T = 1$, 12 and 16. The silhouette score in each title measures how cleanly the three puzzle clusters separate in the original 256-D latent space: at Step 1 the variants of different puzzles overlap heavily ($s \in [0.24, 0.63]$); by Step 12 the recurrence has already injected enough constraint information for the three puzzles to occupy disjoint regions ($s \in [0.33, 0.80]$), and the geometry stays stable through Step 16. For flip and rotation, the 20 variants of each puzzle additionally collapse onto only a handful of canonical representatives, which is what the topological invariance reported in Section 4.7 looks like in latent space.

TR baseline [10] on the same 16×16 Sudoku split using the official open-source implementation with the configuration prescribed in the original paper, and we obtain 84.68% board accuracy under our budget. This gap persists at the same training budget.

CSP-RSB also solves purely non-grid Graph Coloring Problems (GCPs) by replacing the grid topology with the underlying graph's adjacency matrix. We benchmark it against three families of neural CSP baselines on Erdős–Rényi random graphs [41] ($n = 200$ vertices, $k = 3$ colors, edge probability $p = 0.0235$, with a 20% pre-filled hint ratio): dedicated Graph Attention Networks (GATs) [25], the Recurrent Transformer of [10] both in its vanilla form and with our relational bias added in, and the recurrent relational network of [9]. The GAT family (depth up to 10 layers and a 2-layer 16-step weight-tied variant) attains at most 67.05% node accuracy and never attains non-zero

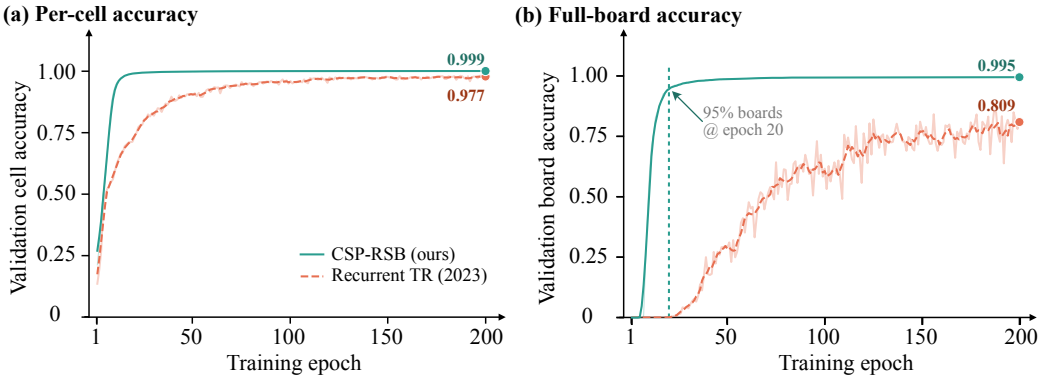

Figure 13: Training trajectories on the RRN-180k Sudoku benchmark, first 200 epochs. (Left) Per-cell validation accuracy: CSP-RSB leaves the cold-start regime within a handful of epochs and saturates near 0.999, while the Recurrent TR baseline plateaus at 0.977 within this window. (Right) Full-board validation accuracy: CSP-RSB crosses the 95% threshold by epoch 20 and asymptotes to 0.995; the Recurrent TR baseline only reaches $\sim$80% at epoch 200 and keeps oscillating, which is consistent with both slower convergence and persistent numerical instability. The final-epoch board accuracy of Recurrent TR over its full 400-epoch budget (Table 1) is 99.51%; the trajectories shown here cover the first 200 epochs of that run, where the convergence-speed gap is most visible.

board accuracy over 200 epochs. The vanilla Recurrent Transformer achieves 37.48% node accuracy because its dense attention lacks a channel for the per-sample adjacency. Adding our relational bias to the same architecture reaches 91.20% node Accuracy and 63.20% Board Accuracy, and RRN reaches 93.95% node accuracy and 76.26% Board Accuracy; CSP-RSB reaches 100% board accuracy in 9 epochs. The full per-variant breakdown is in Table 11.

Table 10: Domain generality. The same recurrent block is retrained on three CSP families with task-specific relation sets: a different constraint family on 10×10 binary puzzles, a larger Sudoku geometry (16×16), and a non-grid graph coloring task on Erdős–Rényi random graphs (n=200, k=3). For binary puzzles and 16×16 Sudoku, we report per-difficulty board accuracy on three test splits (*easy*, *medium*, *hard*) so the previously reported range can be read off directly. For graph coloring, the row labeled "best baseline" is RRN [9], the strongest of the four neural-CSP baselines we trained; the full per-variant breakdown for GAT, vanilla Recurrent TR, Recurrent TR with our relational bias, and RRN is in Table 11.

Domain	**Topology Size**	**Test split**	**Board Accuracy ↑**
Binary Puzzle (10×10)	10×10	easy	100.0%
		medium	100.0%
		hard	99.5%
Sudoku (16×16)	16×16	easy	99.9%
		medium	99.9%
		hard	99.8%
Graph Coloring (CSP-RSB)	n=200, k=3	–	**100%**
Graph Coloring (best baseline)	n=200, k=3	–	76.26% (RRN; 93.95% per-node)

Binary Puzzles (10×10)**.** *Problem definition.* A 10×10 Binary Puzzle is the CSP $< V, D, C >$ with $V = \{\text{cell}_1, \ldots, \text{cell}_{100}\}$ indexed row-major and $D_p = \{0, 1\}$ for every cell. The constraint set C has four families: (i) unary clue constraints $\text{cell}_p = d$ for every pre-filled cell (p, d); (ii) balance constraints, one cardinality constraint per row and per column requiring $\left|\{\text{cell}_\ell = 0 \mid \ell \in S\}\right| = 5$ (equivalently, the same count of 1s) for every row or column index set S of size 10; (iii) triplet constraints, one cardinality constraint per consecutive triple T of three cells along a row or column requiring $\left|\{\text{cell}_\ell = d \mid \ell \in T\}\right| \leq 2$ for each $d \in \{0, 1\}$, which together forbid three identical adjacent values; (iv) row/column uniqueness, requiring that no two completed rows are identical and

no two completed columns are identical. A solution is an assignment that satisfies every constraint in C.

Architectural mapping. The relation set is $\mathcal{R} = \{\text{row}, \text{col}\}$, identical to Sudoku without the block term. The four constraint families above are not added as extra bias terms: only the row/col adjacency enters $\Psi^{\text{BP}}_{i,j} = w_{\text{row}} \cdot \mathbf{1}\{r_i = r_j,\, i \neq j\} + w_{\text{col}} \cdot \mathbf{1}\{c_i = c_j,\, i \neq j\}$, while the balance, triplet, and uniqueness constraints are learned purely from training data. The puzzles are produced by the backtracking generator described in Appendix O, which only certifies that each completed board is a legal solution; mathematical *uniqueness* of the solution is not enforced. The 99.5–100% range in Table 10 is a difficulty range: the trained model is evaluated on three test splits (easy, medium, hard), reaching 100.0% on the easy and medium splits and 99.5% on the hard split. Seed-level variability is reported separately in Table 20 and stays within $\pm 0.10\%$ across the two seeds. The boards the model fails on are therefore typically ambiguous quizzes consistent with multiple completions, not constraint violations.

16×16 **Sudoku.** The architecture is unchanged: $\mathcal{R} = \{\text{row}, \text{col}, \text{blk}\}$ with $\sqrt{N}$=4. We extend E_{val}, E_{row}, E_{col}, E_{blk} to support 16 indices and re-use the same T=16 recurrence schedule. Training uses early stopping with patience 5 epochs, and the number of epochs to convergence depends on the clue count of the split: 113 epochs at 111 clues (*easy*), 112 epochs at 95 clues (*medium*), and 281 epochs at 75 clues (*hard*). Each 16×16 epoch on the RTX 5090 takes $\sim$47 s. The 99.8–99.9% range in Table 10 reads off the three difficulty splits directly: 99.9% on the easy and medium splits and 99.8% on the hard split. Even on the hard split the run stays within 0.1 pt of the easy split, so the larger N does not introduce a per-instance failure mode. The cost shows up instead as a $\sim 2.5\times$ longer training schedule, not as a degraded asymptote.

Erdős–Rényi Graph Coloring. *Problem definition.* Given an undirected graph $G = (\mathcal{V}, E)$ with $|\mathcal{V}| = n$ vertices and k colours, k-Graph Coloring is the CSP $< V, D, C >$ with one variable per vertex, $V = \{v_1, \ldots, v_n\}$ in bijection with $\mathcal{V}$, and a uniform domain $D_i = \{0, 1, \ldots, k-1\}$. The constraint set C has two families: (i) unary clue constraints $v_i = d$ at every hint vertex; (ii) binary inequality constraints $v_u \neq v_v$ for every edge $(u, v) \in E$. A solution is an assignment that satisfies every constraint in C. Each inequality $v_u \neq v_v$ is equivalent to imposing, for every $d \in \{0, \ldots, k-1\}$, the cardinality constraint $\big|\{v_u = d, v_v = d\}\big| \leq 1$, so this domain shares the same cardinality framework as Sudoku.

Architectural mapping. For n=200, k=3, edge probability p=0.0235, and a 20% pre-filled hint ratio, the relation set degenerates to $\mathcal{R} = \{\text{adj}\}$ with $\Psi^{\text{GCP}}_{i,j} = w_{\text{adj}} \cdot \mathbf{1}\{(i, j) \in E\}$, which directly encodes the binary inequality constraints; the colour domain replaces the digit domain in E_{val}. All other components (recurrence depth, QK-Norm, Constraint Projection) are preserved. We train every baseline (GAT, vanilla Recurrent TR, Recurrent TR with our relational bias, RRN) and CSP-RSB for 200 epochs with the same AdamW + OneCycleLR family of schedules; per-method peak learning rate and batch size are listed in the corresponding paragraphs below. Training cost on the RTX 5090 scales near-linearly with graph size: a single epoch takes $\sim$22.5 s at n=200, $\sim$9.4 s at n=100, and $\sim$6.4–6.8 s for $n \in \{20, 50, 64\}$. Only the headline configuration (n=200, two seeds) is reported in Table 10; the smaller graph sizes were used as single-seed development runs. Table 11 reports the full per-variant results, including Node Accuracy.

GAT protocol and failure mode. The three GAT variants in Table 11 share CSP-RSB's optimizer configuration: AdamW with weight decay $1\text{e}{-4}$ and OneCycleLR peaking at $\eta_{\max}$=$1\text{e}{-3}$ (higher than CSP-RSB's $3\text{e}{-4}$ to match the official GAT recipe), 200 epochs, identical 200-node Erdős–Rényi graphs, identical 20% hint ratio, and identical evaluation. The recurrent variant uses weight tying across the 16 recurrent steps, the same as in CSP-RSB. We disable dropout in the GAT layers (the reference implementation does not apply it) and use the full adjacency directly, with no edge sampling, so GAT message passing uses the full constraint graph. Multi-head attention is implemented by feature-dimension expansion rather than head concatenation followed by projection.

The $67.05\%/0\%$ pair in Table 11 has a simple interpretation. For graph 3-coloring, picking a color uniformly at random gives 33.3% node accuracy, and clamping the 20% hint nodes to the ground truth lifts that floor to 46.7% even before any learning. The best GAT variant therefore moves the prediction roughly 20 pt above this trivial floor, but it cannot push the per-graph error rate below the threshold needed for board accuracy: with 80% unknown nodes on a 200-vertex graph, board accuracy requires *all* ~ 160 unknown predictions to be simultaneously correct, so even a 1% residual

Table 11: Erdős–Rényi graph coloring (n=200, k=3, p=0.0235, 20% hints), full breakdown. Three families of neural CSP baselines are compared against CSP-RSB: graph-attention models (GAT), the Recurrent Transformer of [10] both in its vanilla form (no graph hook) and with our relational bias added in, and the Recurrent Relational Network [9]. Among the baselines, RRN (76.26%) and the Recurrent TR with our relational bias (63.20%) are the only two that produce a non-trivial fraction of completely correct graphs; the vanilla Recurrent TR and every GAT variant stay at 0% Board Accuracy within the 200-epoch budget. CSP-RSB converges to perfect Board Accuracy in 9 epochs. Architectural details for the GAT variants are in Appendix O, Table 15.

Model Variant	**Node Accuracy ↑**	**Board Accuracy ↑**
GAT (2-Layer, 1 Recurrence)	56.74%	0.0%
GAT (10-Layer, 1 Recurrence)	66.38%	0.0%
GAT (2-Layer, 16 Recurrences)	67.05%	0.0%
Recurrent TR [10] (vanilla)	37.48%	0.0%
Recurrent TR + our relational bias	91.20%	63.20%
RRN [9]	93.95%	76.26%
CSP-RSB (Ours)	**100.00%**	**100%**

node-level error rate translates into board accuracy on the order of $0.99^{160} \approx 20\%$, and at 33% residual error rate board accuracy is numerically zero. The depth-vs-recurrence pattern inside the GAT block also fits this picture: stacking from 2 to 10 layers gains 9.6 pt at the node level, and replacing depth with 16-step weight-tied recurrence gains another 0.7 pt, but neither change drives the per-graph error rate below the board-accuracy threshold. With the relational bias plus Constraint Projection, CSP-RSB drives the node-level error to zero in this regime, so its node accuracy reaches 100% rather than plateauing in the 60s.

Recurrent Transformer and RRN. The two recurrent neural-CSP baselines pin down two sources of the gap to CSP-RSB: graph awareness and the message-passing structure. We re-implemented the Recurrent Transformer of [10] in its paper-default L1R32H4 configuration with the L_{base} loss (the sum of cross-entropy applied at every (recurrence, block) output, as defined in [10]), and trained it on the same Erdős–Rényi instances. The vanilla model has no channel for the per-sample adjacency: token and positional embeddings carry only the input symbol and the node index, so two graphs with different edge sets but the same partial coloring map to identical inputs. Its node accuracy reaches 37.48%, only 4.2 pt above the 33.3% uniform-color floor and 9.2 pt below the 46.7% hint-clamped floor achievable without any learning. Adding our relational bias to the same Transformer (one learnable scalar gate per head, $w_h \cdot \mathrm{adj}_{i,j} + b_h$, injected into the attention logits) tells the model which node pairs share an edge, and the same architecture reaches 91.20% Node Accuracy and 63.20% Board Accuracy. Per-edge message passing closes most of the remaining gap: RRN, with its LSTM node update, reaches 93.95% Node Accuracy and 76.26% Board Accuracy under the same training budget. The -23.74 pt Board-Accuracy gap from RRN to CSP-RSB reflects three additional priors of CSP-RSB beyond message passing: the hard Constraint Projection on hint nodes, the multi-step SFT schedule, and QK-Norm on a deeply unrolled tied block. Both baselines are trained for 200 epochs with AdamW, OneCycleLR, weight decay $1\mathrm{e}{-4}$, batch size 64 (TR) and 32 (RRN), no augmentation, and the same evaluation protocol as CSP-RSB.

GCP data generation. The Erdős–Rényi instances used by CSP-RSB and by every baseline (GAT, vanilla Recurrent TR, Recurrent TR with our relational bias, RRN) come from the same generator, so the comparison is on a matched distribution. Each instance is built in three steps. First, a ground-truth coloring $\mathrm{GT} \in \{0, \ldots, k{-}1\}^n$ is sampled uniformly at random. Second, for every vertex pair (u, v) with $\mathrm{GT}_u \neq \mathrm{GT}_v$, an edge is added independently with probability p; pairs with $\mathrm{GT}_u = \mathrm{GT}_v$ never become edges, which is what guarantees that GT is a legal k-coloring of the resulting graph. Third, a fraction `hint_rate`$= 0.2$ of vertices is exposed as a clue with their GT color; the remaining 80% are marked as the "uncolored" token k. Mathematical uniqueness of the solution is not enforced: graph k-colorings are invariant under color permutation, and at our edge density, several distinct colorings can satisfy the same constraints. We therefore evaluate against the GT used to generate the graph (cross-entropy on labels, node accuracy as in Table 11), and treat any prediction that respects

every adjacency constraint as a legal solution. In our runs, the model recovers GT exactly, so the two definitions of correctness coincide.

M Auxiliary soft constraint loss

To test whether the model benefits from explicit topological rules encoded as post-hoc penalties, we explored an auxiliary Soft Constraint Loss ($\mathcal{L}_{const}$). All runs in this study use the half-scale Lt-$\frac{1}{2}$ configuration on 10% of RRN-180k. We chose Lt-$\frac{1}{2}$ rather than the headline Standard model because at the Standard width the cross-entropy gradient already drives the row, column, and block sums close to one, leaving no measurable room for an auxiliary marginal penalty to act on. Halving d_{model}, the head count h, and the FFN width brings the model into a regime where the penalty's effect, positive or negative, is visible. The recurrence depth T=16, the optimizer, the multi-step path supervision, and the augmentation policy are inherited from the Standard configuration; the only difference is the width-axis scaling listed in Table 17. In a generalized CSP framework, this loss penalizes deviations of predicted marginal distributions over constrained variable subsets $\mathcal{S}_k$:

$$\mathcal{L}_{const} = \mathbb{E}\left[\sum_{\mathcal{S}_k \in \text{Constraints}} \left\| \sum_{v \in \mathcal{S}_k} P(v) - \mathbf{1} \right\|^2\right]. \tag{11}$$

For Sudoku, the constraint subsets correspond to rows, columns, and blocks:

$$\begin{aligned}\mathcal{L}_{const}^{\text{Sudoku}} = \mathbb{E}_{n,r,k}\left[\left(\sum_{c=1}^{9} P_{n,r,c,k} - 1\right)^2\right] + \mathbb{E}_{n,c,k}\left[\left(\sum_{r=1}^{9} P_{n,r,c,k} - 1\right)^2\right] \\ + \mathbb{E}_{n,b,k}\left[\left(\sum_{(r,c)\in\mathcal{B}_b} P_{n,r,c,k} - 1\right)^2\right],\end{aligned} \tag{12}$$

where $P_{n,r,c,k}$ is the probability that cell (r, c) in puzzle n takes digit k, and $\mathcal{B}_b$ is the index set of the b-th 3×3 block. We add the penalty to the multi-step cross-entropy at every recurrent step, consistent with the path supervision in Section 3, and scale it by a single weight λ. Table 12 sweeps λ at 0, 0.1, and 1.0.

Table 12: Impact of the auxiliary soft constraint loss $\mathcal{L}_{const}$ under the Lt-$\frac{1}{2}$ configuration on 10% of RRN-180k. The cross-entropy is the multi-step path supervision $w_t = t/T$ of Equation (8) applied at every recurrent step in all three runs; only the auxiliary weight λ varies. The penalty hurts at both weights tested.

Constraint loss	**Weight λ**	**Board accuracy ↑**
No (Baseline)	–	**84.10%**
Yes	1.0	76.09%
Yes	0.1	83.26%

With the multi-step schedule held fixed, the penalty has no working point in this regime. At λ=1.0, the auxiliary term dominates the gradient and accuracy drops by 8.01 pt, from 84.10% to 76.09%. Cutting the weight by an order of magnitude to λ=0.1 recovers most of that loss but does not return to the baseline: 83.26% is still 0.84 pt below the unpenalized run, and the direction of every comparison is the same. Adding the penalty hurts, and lowering its weight only hurts less.

We attribute this to the L^2 marginal target itself. The value head already drives the row, column, and block sums of P close to 1 once the predictions sharpen toward a near one-hot posterior (Section 4, Figure 10). The only thing $\mathcal{L}_{const}$ can push against is the residual softness that the cross-entropy is going to remove on its own a step or two later, so a second gradient on the same parameters competes with cross-entropy without adding new information.

This is one penalty form on one constraint family, so we do not generalize to backward-pass penalties as a class. The L^2 marginal target in Equation (11) has no notion of which cells already agree with

the constraint; a Lagrangian schedule that scales the multiplier with the residual violation would be a more principled comparison, and we leave that to future work. For the lightweight 10% RRN-180k regime tested here, we keep $\mathcal{L}_{const}$ off in every other run reported in this paper.

N Optimization stability under deep recurrence

This appendix reports the gradient and attention statistics that support the stability claim in the main text's conclusion. Both CSP-RSB and the Recurrent Transformer baseline of [10] are trained on Given-49k for 200 epochs under the matched setup of Appendix O; the only difference is the architecture itself, so the gap below covers the full design rather than QK-Norm in isolation.

Table 13 and Figure 14 report the two regimes side by side. CSP-RSB ends training with a gradient norm of 0.24 and an attention absolute maximum of 0.34; the Recurrent TR baseline ends at 92.01 and 0.75, with a gradient spike to 149.91 at epoch 180. The 0.34/0.75 contrast is the more informative comparison. The row-wise softmax has $L{=}81$ cell-token keys (after dropping the `CLS` row/column, see the measurement protocol below), so the uniform attention level is $1/81 \approx 0.0123$, and any single row mass approaching 1 means that one query is funneling almost all of its mass into a single key. At 0.75, the Recurrent TR row is therefore $\sim 60\times$ above uniform and within 0.25 of the softmax ceiling, which is the rank-1 regime in which most of the attention information collapses into one token. At 0.34, CSP-RSB sits about $27\times$ above uniform but 0.66 pt below the softmax ceiling, so attention still spreads across topological neighbors rather than collapsing onto a single key. The gradient column points the same direction: a 0.24/92.01 ratio is roughly $390\times$, so the unstabilized baseline carries a gradient signal of an entirely different scale into the optimizer. We therefore attribute the gap to the full architecture, not just to QK-Norm.

Table 13: Gradient and attention stability comparison on Given-49k under the matched setup of Appendix O, using the logged statistics in `figures/data/atts`. The first block reports the final-epoch values (epoch 200) and the second block reports the within-training peak (or epoch-180 spike for Recurrent TR), to make explicit that the comparison mixes final-epoch and peak statistics across the two metric families. CSP-RSB remains stable late in training, while Recurrent TR exhibits much larger gradient norms and sharper attention statistics.

Model	**Epoch**	**Grad. norm**	**Attn. avg. max**	**Attn. abs. max**
CSP-RSB (Ours)	200	0.2358	0.1851	0.3362
Recurrent TR (2023)	200	92.0135	0.3770	0.7450
CSP-RSB (Ours)	Peak observed	3.0358	0.1936	0.3776
Recurrent TR (2023)	Epoch 180 spike	149.9135	0.4028	0.8904

Measurement protocol. The two attention columns of Table 13 are computed from the same per-batch attention tensor $A \in \mathbb{R}^{B\times h\times L\times L}$, where B is the batch, $h{=}8$ is the number of heads, and L is the cell-token count after dropping the `CLS` row and column. We exclude `CLS`–cell entries because their statistics are dominated by the CLS-specific softmax mass and would conflate global aggregation with cell-to-cell constraint propagation; the rest of the paper reports cell-cell attention for the same reason. Within each step, we average across the head dimension first, then compute the two summaries: *Attn. avg. max* is the mean over batch and queries of the row-wise maximum of the head-averaged attention map (how peaked a typical cell's attention is), and *Attn. abs. max* is the absolute maximum of the same head-averaged map across all queries in the batch (the worst-case sharpness reached anywhere in the layer). Statistics are logged every epoch on a fixed validation batch of 256 Given-49k puzzles. We draw the batch once at the start of training and reuse it for every epoch of both runs, so the trajectories in Figure 14 compare the two models on identical inputs. The rest of the setup follows the matched Given-49k configuration of Appendix O, and the curves are plotted directly from the per-epoch logs without smoothing. "Peak observed" in Table 13 is the maximum over all 200 epochs of the corresponding column; for Recurrent TR, the gradient peak coincides with the epoch-180 spike visible in Figure 14.

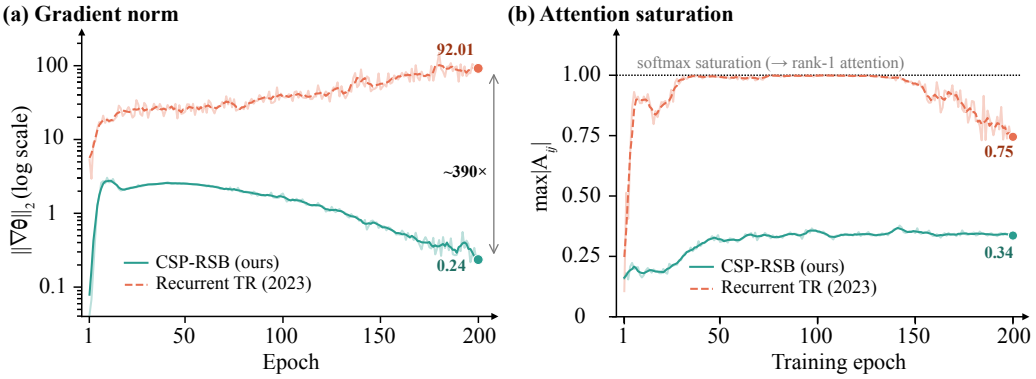

Figure 14: Gradient and attention stability under recurrence. (Left) Gradient norm $\|\nabla\theta\|_2$ on a log scale: CSP-RSB plateaus near 0.24 while the Recurrent TR baseline drifts up to ~ 92, a $\sim 390\times$ gap. (Right) Maximum absolute attention weight $\max|A_{ij}|$: CSP-RSB stays bounded around 0.34, while the Recurrent TR run saturates against the softmax ceiling for most of training, which is the signature of rank-1 attention collapse. The two panels together show that CSP-RSB keeps both the gradient scale and the attention sharpness bounded across deep recurrent training.

O Implementation details and training hyperparameters

This appendix lists every hyperparameter and engineering choice needed to reproduce the experiments in Section 4. Unless explicitly noted, every reported run, across Sudoku (9×9 and 16×16), Binary Puzzles, and Erdős–Rényi Graph Coloring, uses one shared **standard configuration**: only the dataset, the number of training epochs, and the batch size differ across experiments, while model width, depth, recurrence steps, optimizer, learning-rate schedule, and regularization are held fixed.

The standard configuration is described in the next two paragraphs. Two ablation studies deliberately depart from it: the Constraint Projection ablation in Section 4.4 (Table 4) uses a quarter-scale model (Lt-$\frac{1}{4}$) to amplify the role of architectural priors under extreme data scarcity, and the auxiliary soft-constraint-loss study in Appendix M (Table 12) uses a half-scale model (Lt-$\frac{1}{2}$) so that the auxiliary penalty has measurable room to act on. Table 17 lists both lightweight variants alongside the standard configuration. Every other table, figure, and result in the paper uses the standard configuration unless this paragraph explicitly says otherwise.

Architecture. CSP-RSB repeats a 2-layer Transformer block for $T{=}16$ tied steps. The block has hidden width $d_{\text{model}}{=}256$, $h{=}8$ attention heads for the Sudoku configurations (head dim $d_h{=}d_{\text{model}}/h{=}32$), and feed-forward width $d_{\text{ff}}{=}512$. The Binary Puzzle run uses $h{=}4$ heads (head dim 64) to match the reference recipe distributed with the Binary Puzzle generator; see Table 16 for the cross-domain settings. The non-linearity in the feed-forward sub-layer is GELU. QK-Norm is applied to Q and K independently using LayerNorm before the scaled dot product. Type embeddings have cardinality $|\{\text{fixed}, \text{mutable}\}|{=}2$. Structural positional embeddings use one learnable table per relational axis: $E_{\text{row}}, E_{\text{col}}, E_{\text{blk}}$ for Sudoku ($N{\in}\{9, 16\}$), $E_{\text{row}}, E_{\text{col}}$ for Binary Puzzles, and a single E_{node} for Graph Coloring. The relational bias Ψ uses one learnable scalar w_k per relation type, broadcast across heads; the Standard `CLS`-bearing configuration additionally uses a learnable w_{CLS} for `CLS`-cell interactions. The reported Standard checkpoint stores approximately 1.88M tensor entries, including buffers; Table 18 gives the per-module storage breakdown, not a trainable-parameter count.

Optimization. We use AdamW with parameter-group weight decay: weight matrices in the encoder and actor receive a decay of $1\text{e}{-4}$, while biases, LayerNorm scales, embedding tables, and the relational-bias scalars $\{w_k\}$ are exempt from decay (decay 0) to preserve the soft logical priors. The learning rate is shaped by a OneCycle schedule [45]: peak learning rate $\eta_{\max}{=}3\text{e}{-4}$, initial learning rate $\eta_{\max}/25{=}1.2\text{e}{-5}$, 10% of total optimizer steps used for cosine warm-up, and a final learning rate of $\eta_{\max}/(25\cdot10^3){=}1.2\text{e}{-8}$ via cosine annealing. Dropout is 0.1 on residual connections and attention. We use the same optimizer configuration for all datasets, ablation runs, and lightweight runs

reported in this paper; only the number of training epochs and batch size vary across experiments, as listed in Table 14.

Training schedule. Per-experiment epoch counts and batch sizes are summarized in Table 14. The multi-step path supervision uses linear weights $w_t{=}t/T$ as in Equation (8); we did not tune this curriculum.

Table 14: Per-experiment training schedule. The "Config" column distinguishes the standard configuration (Std) from the two lightweight variants: Lt-$\frac{1}{2}$ for the auxiliary soft-constraint-loss study (App. M) and Lt-$\frac{1}{4}$ for the Constraint Projection ablation (Section 4.4). Table 17 defines all three variants, which share the same OneCycle AdamW schedule. Every Sudoku, Aux, and Constraint-Projection-ablation run uses batch size 256; the Binary Puzzle run is the only exception, using batch size 128 to match the reference recipe distributed with the Binary Puzzle generator (the same value is cross-listed in Table 16). Wall times are wall-clock single-seed durations on a single RTX 5090; values marked $\sim$ are estimated from logged epoch times rather than measured end-to-end, with deviations expected to be within $\pm$10–20%. ‡ The $16{\times}16$ Sudoku run uses early stopping (patience 5 epochs) and converges in 112–281 epochs depending on the clue count of the test split (e.g., 112 epochs for 95-clue, 113 for 111-clue, 281 for 75-clue).

Experiment	Config	Epochs	Batch size	Wall time
RRN-180k (full, 180k)	Std	400	256	$\sim$4.7 h
RRN-180k (10%, 18k)	Std	400	256	$\sim$1.7 h
Given-49k	Std	100	256	$\sim$1.2 h
Binary Puzzle ($10{\times}10$)	Std	400	128	$\sim$5.1 h
$16{\times}16$ Sudoku‡	Std	112–281	256	$\sim$1.5–3.7 h
Graph Coloring ($n{=}200$)	Std	200	256	$\sim$1.25 h
RRN-180k (5%, 9k), CP ablation	Lt-$\frac{1}{4}$	200	256	$\sim$0.4 h
RRN-180k (2.5%, 4.5k), CP ablation	Lt-$\frac{1}{4}$	200	256	$\sim$0.2 h
Aux soft-constraint loss (10%, 18k, App. M)	Lt-$\frac{1}{2}$	200	256	$\sim$0.9 h

Data preprocessing. Sudoku boards are flattened into row-major sequences of length N^2. Empty cells use the reserved token 0, and clue cells use tokens 1 to N. The binary mutability mask M is built from the input, with $M_i = 1$ when $x_i = 0$. Data augmentations (rotations by 0°, 90°, 180°, and 270°; horizontal and vertical flips; and digit permutations from the symmetric group S_N) are sampled uniformly within each batch when enabled. No augmentations are applied at test time. For Graph Coloring, we sample a new Erdős–Rényi graph for each training instance using the parameters in Appendix L, with hint nodes drawn uniformly and without replacement.

Inference. At evaluation we run the same $T{=}16$ recurrence used in training for all headline accuracy results. Predictions are taken as $\arg\max$ over the value head; we do not use beam search or sampling. Constraint Projection is applied at every step, including the final one, so clue cells are correct by construction.

Baselines. We retrained RRN [9], Recurrent TR [10], and Causal LM (Shah et al., 2024) [3] on both the RRN-180k and Given-49k datasets, using the official open-source implementation of each method with the default hyperparameters from their respective repositories. We only changed the dataset path so each baseline in Table 1 was trained on the same data and evaluated on the same Kaggle splits as CSP-RSB. We also reproduced the reference value of 84.68% for the 16×16 Sudoku case in Recurrent TR, used in Appendix L, under the same retraining protocol. We used the official Recurrent TR codebase with the configuration described in the original paper for 16×16 Sudoku. The original study said computational limitations prevented them from scaling this experiment further, so we kept this comparison within our budget rather than copying a previously published table. We did not retrain the only baseline, Trial & Error [11]; that row reproduces the original publication's numbers and appears only in the RRN-180k section of Table 1, since the original paper does not report results on Given-49k. For the GAT baseline in graph coloring, we used PyTorch Geometric with the depth and recurrence configurations of Table 10 and the per-variant breakdown in Table 11. The hyperparameters for each configuration are listed in Table 15.

Table 15: GAT baseline configurations on Erdős–Rényi Graph Coloring. The three variants share every entry except spatial depth and the number of recurrent applications; the recurrent variant uses weight-tied recurrence in the same sense as CSP-RSB. The optimizer block follows the official GAT reference implementation, so the peak learning rate ($1e{-3}$) is one notch above CSP-RSB's $3e{-4}$. We keep the GAT-recommended value to avoid penalizing the baseline with a schedule that was never tuned for it. Edge sampling is disabled, so message passing always operates on the full adjacency.

Hyperparameter	**Value**
Architecture (per variant)	
GAT (2-Layer, 1 Recurrence)	layers $= 2$, recurrences $T = 1$
GAT (10-Layer, 1 Recurrence)	layers $= 10$, recurrences $T = 1$
GAT (2-Layer, 16 Recurrences)	layers $= 2$, recurrences $T = 16$ (weight-tied)
Hidden width / heads	256 / 8
Head mapping	identity (per-head feature expansion, no concat-then-project)
Output aggregation	mean over recurrence steps, then linear class head
Dropout	0.0 (not applied in the reference `RecurrentGAT.forward`)
Optimization (shared across variants)	
Optimizer	AdamW
Peak learning rate $\eta_{\max}$	$1e{-3}$
LR schedule	OneCycleLR (warm-up + cosine annealing)
Weight decay	$1e{-4}$
Epochs / batch size	200 / 256
Edge sampling	none (full adjacency $\rightarrow$ `edge_index`)

Binary puzzle and graph coloring data and optimizer settings. The two cross-domain experiments use the same backbone as the Sudoku runs but with smaller training corpora; Binary Puzzles additionally use $h{=}4$ heads (instead of 8) and a smaller batch size to match the reference recipe distributed with the Binary Puzzle generator. Table 16 summarises the dataset sizes and per-task hyperparameters not captured by the shared standard configuration. For Binary Puzzles, the generator (`BinaryPuzzleGenerator`) builds each board by backtracking under the balance and triplet constraints and discards boards whose rows or columns duplicate, then applies a uniform `hole_rate` mask to expose the quiz; legality of the completed board is verified by a separate `validate_binary_puzzle` routine, while mathematical solution uniqueness is not separately certified. For GCP, the generator is the procedure described in Appendix L: a random coloring GT is drawn first, edges are then added only between vertices with different colors so that GT is guaranteed legal, and a 20% hint mask is applied. In both cases, the loss is cross-entropy restricted to the unknown positions (mask-only).

Lightweight configurations (ablation studies only). Two ablations are run at smaller widths than the headline Standard model so that the component being studied is not absorbed by surplus capacity. The Constraint Projection study (Section 4.4, Table 4) runs on 5% and 2.5% of RRN-180k and uses Lt-$\frac{1}{4}$, with d_{model}, the head count h, and the FFN width scaled to a quarter of the Standard values; at full Standard width, the over-parameterized network would substitute for the projection operator. The auxiliary soft-constraint-loss study (App. M, Table 12) runs on 10% of RRN-180k and uses Lt-$\frac{1}{2}$, with the same three dimensions halved. Except for the width axis, both variants match the Standard run: same recurrence depth $T{=}16$, same optimizer, OneCycle schedule, multi-step path supervision, QK-Norm, augmentation policy, and inference protocol. Table 17 lists the three configurations side by side.

Checkpoint storage and learned relational weights. Table 18 records stored tensor entries in the reported checkpoint, including buffers; tensor dtype does not establish trainability. The bias generator contains 19,688 entries, of which 19,683 are gradient-free constants and five are learned scalars. The approximately 1.88M stored entries must therefore not be interpreted as a trainable-parameter count. The separate 243 integer index entries are also buffers. This distinction separates added storage from learned capacity when interpreting the bias's gains.

Table 16: Binary Puzzle and GCP data and optimizer settings used by the cross-domain experiments in Appendix L. "Standard" in the architecture row refers to the configuration listed in Table 17 (Std column), with the relation set as noted in the text. Binary Puzzles use 4 attention heads (head dim 64) and batch size 128 to match the reference recipe distributed with the Binary Puzzle generator; everything else is shared with Sudoku.

Setting	**GCP** (n=200, k=3)	**Binary Puzzle** (10×10)
Dataset sizes		
Train	10,000	9,000
Validation	1,000	1,000
Test	1,000	1,000
Architecture		
Backbone	Standard (d_{model}=256, 2 layers, T=16)	Standard (d_{model}=256, 2 layers, T=16)
Attention heads	8	4
Relation set $\mathcal{R}$	$\{\text{adj}\}$	$\{\text{row}, \text{col}\}$
Optimization		
Optimizer	AdamW	AdamW
Peak LR $\eta_{\max}$	3e$-$4 (OneCycleLR)	3e$-$4 (OneCycleLR)
Weight decay	1e$-$4 (no decay on bias / LN / embeddings)	1e$-$4 (no decay on bias / LN / embeddings)
Epochs	200	400
Batch size	256	128
Loss	cross-entropy on unknown nodes	cross-entropy on masked cells

Table 17: Architecture variants used in this paper. Std is the headline configuration used by every Sudoku run in the main text and by GCP in Table 10; Lt-$\frac{1}{2}$ is used only for the auxiliary soft-constraint-loss study (App. M, Table 12); Lt-$\frac{1}{4}$ is used only for the Constraint Projection ablation (Section 4.4, Table 4). The Binary Puzzle run shares the Std width but uses h=4 heads instead of 8 (Table 16). The optimizer, schedule, recurrence depth, and batch size are shared across all three variants; only the width-axis dimensions differ. The Std checkpoint-entry count includes buffers (Table 18). No corresponding stored-entry count is reported for the lightweight variants; their architectural dimensions are specified explicitly.

Hyperparameter	**Standard (Std)**	**Lt-$\frac{1}{2}$ (Aux)**	**Lt-$\frac{1}{4}$ (CP)**
d_{model}	256	128	64
Attention heads h	8	4	2
Head dim d_h	32	32	32
FFN width d_{ff}	512	256	128
Layers (per recurrent step)	2	2	2
Recurrence steps T	16	16	16
Checkpoint entries (approx.)	1.88 M	–	–
Peak learning rate $\eta_{\max}$	3e$-$4	3e$-$4	3e$-$4
Batch size	256	256	256

P Statistical significance and seed variability

Headline CSP-RSB Sudoku results use three seeds $\{0, 1, 42\}$ (Table 19); the CP-SAT reference uses a fixed solver configuration, not a three-seed neural experiment. The original structural ablations, data-efficiency and augmentation studies, depth/recurrence sweep, CP controls, Binary Puzzle, and headline $n = 200$ graph-coloring result use two seeds $\{0, 1\}$ (Table 20). The 16×16 Sudoku runs use one seed. The matched-backbone ladder and each corrupted-topology retraining setting use one seed; the additional graph configurations are single-seed development runs except for the reused two-seed headline row. Single-seed results have no across-seed uncertainty estimate. Standard deviations for two- and three-seed results are population standard deviations, not standard errors or confidence intervals, and reflect seed variation rather than data-resampling uncertainty.

Table 18: Stored checkpoint entries grouped by module (epoch 171). Counts include constant buffers; the `float32` subtotal is not a trainable-parameter subtotal. Of the 19,688 bias-generator entries, 19,683 are constants and five are learned scalars. Shares refer to the full stored-entry total.

Module	**Stored entries**	**Share**
Reasoning block (`encoder.reasoning_block`)	1,579,776	84.20%
Actor / value head (`actor.shared_net` + `actor.fill_head`)	265,225	14.13%
Relational bias generator (`encoder.bias_gen`)	19,688	1.05%
Structural embeddings (row, column, block; 9×256 each)	6,912	0.37%
Value embedding (`encoder.val_emb`; 10×256)	2,560	0.14%
LayerNorms (`input_norm` + `output_norm`)	1,024	0.05%
Type embedding + `CLS` token	768	0.04%
Floating-point subtotal (including buffers)	1,875,953	99.99%
Constant indices (`row_idx`, `col_idx`, `blk_idx`; `int64` buffers)	243	0.01%
Checkpoint total	**1,876,196**	100%

Table 19: Sudoku board accuracy with seed variability (CSP-RSB rows only). Values are mean$\pm$std over the three seeds $\{0, 1, 42\}$ with otherwise identical hyperparameters; the standard deviation is the population standard deviation across these three runs. The Kaggle columns follow Table 1: *Kaggle Unfiltered* is the full 3M-puzzle dataset of [18] and *Kaggle Filtered* is the 1.9M-puzzle subset curated by [3] that retains only puzzles solvable by seven human Sudoku strategies.

Training Data	**Model**	**Acc (RRN Test)**	**Acc (Kaggle Unfiltered)**	**Acc (Kaggle Filtered)**
RRN-180k	CSP-RSB (Ours)	99.94 ± 0.03 %	99.13 ± 0.33 %	99.78 ± 0.02 %
Given-49k	CSP-RSB (Ours)	99.84 ± 0.07 %	97.81 ± 0.23 %	99.27 ± 0.12 %

Q Computing resources

Neural-model experiments and profiling use a single NVIDIA RTX 5090 (32 GB), with no multi-GPU or multi-node training, on a host allocated 25 vCPUs from an Intel Xeon Platinum 8470Q. Per-run training times are in Table 14. The CP-SAT reference was measured separately on an Intel Core i5-12600KF under Windows 10, using one worker and Python 3.12.14. Its times are raw local measurements, with no CPU-performance conversion or cross-hardware speedup claim.

The original neural experiments accounted for approximately 165 GPU-hours, excluding the additional matched-backbone and corruption runs and the CPU-only reference, aggregated across the original tables and seeds, broken down approximately as follows: (i) main Sudoku results in Table 1 (3 seeds for the CSP-RSB rows on RRN-180k and Given-49k; $\sim$18 h), (ii) data-efficiency sweeps in Table 3 (2 seeds for the 10%/30%/50% ratios, with the 100% endpoint reusing the 3-seed RRN-180k run from (i); $\sim$17 h), (iii) augmentation study in Table 8 (2 seeds per row; $\sim$7 h), (iv) structural ablations in Table 2 (2 seeds) and the depth/recurrence sweep in Table 6 (2 seeds; together $\sim$55 h, the largest single component), (v) Constraint Projection ablations in Tables 4 and 5 (2 seeds each; Lt-$\frac{1}{4}$ ablation $\sim$3 h, no-type-embedding control $\sim$5 h), (vi) auxiliary soft-constraint-loss study in Table 12 (1 seed, Lt-$\frac{1}{2}$ configuration; $\sim$5 h), (vii) stability tracking in Appendix N ($\sim$5 h, recorded inside the main Sudoku runs), and (viii) cross-domain transfer in Table 10 (Binary Puzzle, 16×16 Sudoku, and Erdős–Rényi Graph Coloring at n=200 and smaller sizes; $\sim$30 h). The full project, including failed runs and exploratory configurations not reported here (notably depth/recurrence trade-off scans beyond Table 6, early QK-Norm/no-QK-Norm comparisons, additional graph sizes used for development, and several lightweight-configuration sweeps that were superseded by the runs above), needed roughly $1.5\times$–$2\times$ the compute of the reported runs (an additional 75–150 GPU-hours).

R Controlled component additions on a matched backbone

To separate the effects of input representations from those of the recurrent update, we compare six configurations of the same recurrent Transformer on the 10% subset of RRN-180k (18,000 training puzzles). All runs share the width, depth, optimizer, learning-rate schedule, data split, and 400-epoch

Table 20: Seed variability for the secondary experiments. "Seeds" is the number of independent runs averaged for that row (2 = $\{0, 1\}$, 1 = single run). Rows marked "–" in the rightmost (standard-deviation) column had only a single run, by design rather than by oversight: these are exploratory sweeps where the trend, not the absolute value, is the claim. Rows above the divider use the Std configuration. Rows below the divider belong to the Constraint Projection control studies and are split into two groups: the Lt-$\frac{1}{4}$ ablation behind Table 4 and the no-type-embedding control study behind Table 5, both run with two seeds. Rows where both seeds saturated to the same accuracy (100% for CSP-RSB on Graph Coloring; 0.0% for the GAT baseline) report a measured std of 0.00%. †The 100% (180k) data-efficiency cell reuses the three-seed RRN-180k headline run reported in Table 19 (seeds $\{0, 1, 42\}$); we list it here so the data-efficiency sweep has a consistent endpoint at 100% data, not as an extra run. ‡The 10% (18k) data-efficiency cell shares the same two-seed run as the Augmentation "18k, no Aug" row above it; we list it in both blocks so each table caption can be cross-checked without back-references.

Experiment	Cfg	Setting	Mean	Seeds	Std
Data efficiency (Tab. 3)	Std	10% (18k)‡	84.10%	2	1.20%
Data efficiency	Std	30% (54k)	99.32%	2	0.08%
Data efficiency	Std	50% (90k)	99.71%	2	0.04%
Data efficiency	Std	100% (180k)†	99.94%	3	0.03%
Augmentation (Tab. 8)	Std	18k + Aug	99.80%	2	0.07%
Augmentation	Std	18k, no Aug	84.10%	2	1.20%
Structural ablation (Tab. 2)	Std	Given-49k, w/o Learnable Bias	99.77%	2	0.03%
Structural ablation	Std	Given-49k, w/o Soft Bias (Hard Bias only)	98.68%	2	0.11%
Structural ablation	Std	Given-49k, w/o Pos Embedding	99.73%	2	0.02%
Structural ablation	Std	Given-49k, w/o Type Embedding	99.63%	2	0.09%
Structural ablation	Std	Given-49k, w/o CLS Token	99.84%	2	0.04%
Depth/recurrence (Tab. 6)	Std	1 layer / 16 recurrences	56.84%	2	0.52%
Depth/recurrence	Std	2 layers / 16 recurrences	62.29%	2	0.31%
Depth/recurrence	Std	4 layers / 8 recurrences	47.61%	2	0.67%
Depth/recurrence	Std	8 layers / 4 recurrences	20.47%	2	0.77%
Depth/recurrence	Std	16 layers / 2 recurrences	0.56%	2	0.03%
Depth/recurrence	Std	32 layers / 1 recurrence	0.60%	2	0.05%
Graph coloring (Tab. 10)	Std	CSP-RSB, n=200, k=3	100%	2	0.00%
Graph coloring	Std	GAT (best variant)	0.0% (board)	2	0.00%
Binary Puzzle (Tab. 10)	Std	10×10	99.85%	2	0.10%
16×16 Sudoku (Tab. 10)	Std	3 difficulty splits, 1 seed each	99.8–99.9%	1	–
Constraint Projection (Tab. 4)	Lt-$\frac{1}{4}$	5% (9k), w/ CP	80.59%	2	0.90%
Constraint Projection	Lt-$\frac{1}{4}$	5% (9k), w/o CP	58.47%	2	1.50%
Constraint Projection	Lt-$\frac{1}{4}$	2.5% (4.5k), w/ CP	70.14%	2	0.80%
Constraint Projection	Lt-$\frac{1}{4}$	2.5% (4.5k), w/o CP	40.12%	2	1.80%
Constraint Projection (Tab. 5)	Std, no E_{type}	10% (18k), w/ CP	85.47%	2	0.17%
Constraint Projection	Std, no E_{type}	10% (18k), w/o CP	79.20%	2	0.23%
Constraint Projection	Std, no E_{type}	5% (9k), w/ CP	62.40%	2	0.36%
Constraint Projection	Std, no E_{type}	5% (9k), w/o CP	36.74%	2	0.44%

budget. Table 21 reports one seed per configuration. These controlled additions complement the component removals in Table 2; they are a separate experiment, rather than additional seeds for that table.

Table 21: Matched-backbone component additions on RRN-18k. Each row reports a single-seed board accuracy. R2p branches from R1, so the rows do not form a strictly cumulative sequence.

ID	Configuration	Board accuracy (%)
R0	Recurrent Transformer (vanilla)	50.28
R1	R0 + composite and type embeddings	55.83
R2	R1 + relational bias	75.96
R2p	R1 + constraint projection	60.43
R3	R1 + relational bias + constraint projection	80.96
R4	R3 + QK-Norm	84.16

The embeddings account for a gain of 5.55 percentage points (R0 to R1). Holding those embeddings fixed, adding relational bias gives 20.13 points (R1 to R2), while projection alone gives 4.60 points (R1 to R2p). Adding projection to the bias-equipped model gives another 5.00 points (R2 to R3), and QK-Norm adds 3.20 points (R3 to R4). Thus, the 8.20-point difference between R2 and R4 is not an

input-embedding effect. These results support distinct contributions from topology, repeated axiom preservation, and attention stabilization in this low-data setting. As single-seed comparisons, they do not establish the variance or statistical significance of each marginal gain.

S Retraining with corrupted relational topology

The relation types supplied to CSP-RSB are specified by hand; only their weights are learned. To examine sensitivity to an incorrect supplied topology, we retrain the model from scratch using corrupted Sudoku relation graphs. Each of the row, column, and block relations contains 324 undirected edges, with eight neighbors per cell. For each relation, we replace a fraction ρ of true edges with false edges using degree-preserving double-edge swaps. This preserves each cell's degree while changing its partners. We share a single corrupted graph at each ratio across training instances, giving a fixed misspecified prior rather than per-instance noise.

The study uses the RRN-18k setting of Table 2, with one training seed per ratio and a 400-epoch budget. The three relation weights are initialized to 0.25. Table 22 reports the learned weights and board accuracy for the soft additive bias; the hard-mask comparison is evaluated at seven selected ratios. The ratio $\rho = 0$ recovers the uncorrupted topology. This experiment measures adaptation through retraining with an incorrect graph and does not measure a trained model's response to a sudden deployment-time topology change.

Table 22: Corrupted-topology retraining. Accuracy columns are board accuracy in percent. "Mean/init" is the mean learned relation weight divided by its initial value of 0.25, reported from the experiment; displayed weights are rounded. Dashes indicate hard-mask settings that were not evaluated. Each ratio uses one seed.

ρ	w_{row}	w_{col}	w_{blk}	Mean/init	Soft bias	Hard mask
0.00	1.33	1.32	0.59	$4.32\times$	84.11	62.51
0.01	1.05	1.04	0.42	$3.35\times$	78.65	50.76
0.02	0.94	0.95	0.39	$3.05\times$	76.36	–
0.03	0.77	0.79	0.30	$2.49\times$	73.11	–
0.04	0.63	0.59	0.17	$1.86\times$	70.98	–
0.05	0.47	0.49	0.10	$1.41\times$	69.28	23.64
0.06	0.42	0.41	0.08	$1.22\times$	68.17	–
0.07	0.35	0.33	0.07	$0.99\times$	67.18	–
0.08	0.26	0.27	0.06	$0.79\times$	66.69	–
0.09	0.22	0.23	0.05	$0.65\times$	65.63	–
0.10	0.19	0.20	0.06	$0.61\times$	64.79	8.63
0.25	0.05	0.03	0.02	$0.14\times$	64.19	0.48
0.50	0.02	0.01	0.01	$0.06\times$	63.78	0.00
1.00	0.02	0.01	0.01	$0.05\times$	63.59	0.00

The learned weights are large for the correct topology and are reduced as corruption increases. At $\rho = 1$, soft-bias accuracy is 63.59%, close to the 63.68% bias-free result in Table 2, whereas hard-mask accuracy is 0.00%. An additive bias can approach zero without deleting attention connections; a hard mask can exclude useful connections altogether. The observed return toward bias-free performance is an empirical result for these graphs and retraining runs, not a guaranteed accuracy floor for arbitrary incorrect topologies. It also does not remove the need to specify the relation set or establish that the model can discover missing relations.

T Diagnostics on Sudoku with multiple feasible solutions

The headline Sudoku benchmarks use a single reference solution per puzzle. To examine behavior when several completions are feasible, we also evaluate 500 ambiguous instances derived from RRN test puzzles. Each instance has between two and thirteen solutions, exhaustively enumerated and verified with CP-SAT. This diagnostic is separate from the full-dataset CP-SAT reference evaluation.

At positions where feasible solutions disagree, the entropy diagnostic is 0.128, compared with 0.693 for a uniform split between two candidates. Moreover, 99.98% of predicted digits belong to the set of values occurring in at least one feasible solution at the corresponding cell. This measures agreement

with per-cell feasible-value sets, not whole-board validity: the feasible solution witnessing one cell's value may differ from the solution witnessing another cell's value.

Indeed, some instances contain per-cell choices that do not jointly form a consistent solution. Concentrated local predictions therefore do not certify a valid joint assignment, and these diagnostics do not establish how several solutions are represented in hidden states. We do not claim to solve selection among multiple feasible completions; this coordination failure remains a limitation of independent value-head decoding.

U Graph-coloring configurations across sizes and parameters

Table 23 expands the graph-coloring results in Appendix L. The same architectural relation template, $\mathcal{R} = \{\mathrm{adj}\}$, is used throughout, with each graph's adjacency supplied as input. The relation set is not returned across the listed graph sizes, edge probabilities, or hint ratios. Models are trained for the respective configurations; these results do not represent zero-shot size generalization by a single fixed checkpoint.

Table 23: Graph-coloring configurations. The first seven rows are single-seed development runs; the final row reuses the two-seed headline configuration rather than adding an independent experiment. Accuracy is measured at the full-board level.

Vertices n	**Edge prob. p**	**Hints**	**Board acc. (%)**	**Ep.→95%**	**Seeds**
20	0.2000	0.40	99.9	12	1
50	0.1200	0.45	100.0	8	1
50	0.1500	0.20	99.3	21	1
64	0.0235	0.25	100.0	11	1
100	0.1000	0.40	100.0	14	1
100	0.0470	0.25	100.0	5	1
100	0.0420	0.12	100.0	8	1
200	0.0235	0.20	100.0	9	2

Across these configurations, board accuracy ranges from 99.3% to 100%, and the 95% threshold is reached within 5–21 epochs. These runs provide evidence that one adjacency relation template is usable over the listed settings. The limited seed coverage and development-run status of the first seven rows constrain conclusions about variability or generalization to untested graph distributions.

V Inference costs and comparison with test-time search

Measured neural inference. We profile the Standard CSP-RSB model with $T = 16$ on one RTX 5090 in FP32, with TF32 disabled. Latency is the median of 200 runs after warm-up. The forward computation is approximately 4.3 GFLOPs per board; analytic FLOPs were cross-checked with PyTorch's counter. Table 24 separates single-board latency from amortized batch throughput. The latter is not the latency experienced by an isolated board. The Standard FFN width is 512, as in Table 17.

Table 24: Measured CSP-RSB inference on RTX 5090. Timing uses FP32 with TF32 disabled, after warm-up. Batch latency is reported per board; memory is the reported memory usage for that batch.

Batch size	Latency per board	Throughput	Memory
1	7.08 ms	–	17.4 MiB
256	0.172 ms (amortized)	5,827 boards/s	283 MiB

The relational-bias generator stores $19{,}688$ entries, but $19{,}683$ are gradient-free constants, and only five are learned scalars. Recomputing the bias at each step adds 2.8% relative to the no-bias profiling condition; constraint projection adds 0.7%. These measurements describe implementation overhead, not improvements in asymptotic complexity. Table 18 distinguishes checkpoint storage from trainable parameters.

Analytical accounting for Trial & Error. Trial & Error [11] uses autoregressive test-time search. We retain its published accuracy in Table 1; we do not truncate its evaluation to match CSP-RSB's fixed 16-step schedule. For a 17-clue puzzle, the no-backtracking transcript considered here has 65 tokens; 500 tokens is the cap in the released implementation, not a universal budget for the paper's published accuracy. The released implementation recomputes the prefix, leading to approximately quadratic growth with transcript length. Table 25 gives analytical estimates for that implementation and a hypothetical ideal key–value cache. These are neither measured GPU latencies nor a newly evaluated cached solver.

Table 25: Analytical Trial & Error FLOPs per board. The ideal-cache column is a hypothetical implementation estimate. CSP-RSB uses a fixed 16-step forward computation of approximately 4.3 GFLOPs. Hardware-dependent speedup claims are not inferred from this table.

Transcript length	Released implementation	Ideal KV cache
65 tokens	283 GFLOPs	7.2 GFLOPs
500 tokens	11,492 GFLOPs	45.3 GFLOPs

W Non-learning CP-SAT reference

Model and inputs. We use OR-Tools CP-SAT [19], package version 9.15.6755, adapting the official Sudoku example at commit `551ad10d94835c99e5e1e684500d3db398c0e345`. Each board has 81 integer variables in $\{1, \ldots, 9\}$, 27 `AllDifferent` constraints for rows, columns, and boxes, and equality constraints for the given digits. There is no optimization objective, learned model, solution hint, or use of the reference answer during solving. We seek one feasible completion, not an enumeration or uniqueness proof.

The evaluation covers all 18,000 supplied RRN Test puzzles and all 3,000,000 puzzles in the Radcliffe Kaggle release, version 2 [18]. The RRN file has no header, and we retain its first row. The two strategy-filtered files distributed with Shah et al. [3] contain 1,804,463 and 100,000 puzzles, totaling 1,904,463. Exact matching verified that every filtered puzzle and its reference answer occur in the unfiltered corpus, with no duplicate puzzle keys. We therefore solve RRN Test and Unfiltered once each and derive Filtered statistics using the fixed instance-ID mapping. This entails 3,018,000 solver calls; the filtered rows are not additional independent calls.

Fixed execution and scoring. Each puzzle uses one worker, solver seed 1, and a 10-second solve-time limit, with all other settings at the fixed version's defaults. Boards are processed sequentially. We measure on an Intel Core i5-12600KF under Windows 10 with Python 3.12.14. This CPU differs from the neural-model host: we report its actual measurements without scaling by CPU benchmark scores or forming a CPU-versus-GPU speedup ratio.

Board accuracy uses the same reference-match definition as the neural results: all original blank cells must match the supplied answer. Separately, we verify all row, column, and box constraints and preservation of every given. Unresolved or infeasible cases would remain in both the accuracy denominator and timing population. The primary wall-clock measurement includes model construction, solving, and output decoding, but excludes input reading, independent verification, and database writes. Solver-reported wall time is retained separately. Table 26 reports statistics over all instances in each column.

Table 26: CP-SAT full-corpus results on i5-12600KF, one worker. Accuracy and validity are separate checks. Time includes construction, solve, and decoding; Filtered inherits the times of its mapped Unfiltered instances.

Dataset	Puzzles	Accuracy / valid	Mean (ms)	Median (ms)	P95 (ms)
RRN Test	18,000	100%/100%	2.240	2.018	3.723
Kaggle Unfiltered	3,000,000	100%/100%	3.230	3.127	5.086
Kaggle Filtered	1,904,463	100%/100%	2.815	2.366	4.541

All evaluated instances returned a feasible completion (reported as `OPTIMAL` for these satisfaction models), matched the reference, and passed independent validity checks. There were no unresolved, infeasible, or invalid cases and no valid-but-reference-mismatched outputs. All 3,018,000 saved predictions were checked against the original inputs and labels; an independent database and mapping audit reproduced the reported counts and timing statistics. The complete run, including reading, verification, storage, and aggregation, took approximately 2.86 wall-clock hours. The non-learning row demonstrates the distinction between explicit symbolic solving and learned inference; it is not evidence of a training-data requirement for CP-SAT.

X Asset licenses, code release, and credits

Code availability. The CSP-RSB implementation, training scripts, and Given-49k construction pipeline were supplied as an anonymized supplementary archive at submission. A public repository URL is not yet available. The existing implementation is distributed under the MIT License. The CP-SAT reference uses the separate version and evaluation protocol in Appendix W; we retained input hashes, per-instance predictions, and environment metadata for verification.

Software environment. The `requirements.txt` shipped with the release specifies the neural-model environment. The CPU-only CP-SAT reference uses the separate environment in Appendix W. The base stack is PyTorch ≥ 2.10 with CUDA 12.6; `torchvision` and `torchaudio` are pulled in for completeness but are not used by the training scripts. The Graph Coloring baseline additionally needs `torch-geometric` (≥ 2.5); on CUDA 12.6, the optional companions `torch-scatter` and `torch-sparse` may need to be built from source rather than installed from wheels, and the `README.md` flags this. Data construction and verification rely on `numpy` ($\geq 1.26.4$), `pandas` (≥ 2.1) and `scikit-learn`, and the training entry-points use `tyro` ($\geq 0.8.4$), `tqdm`, `matplotlib` and `einops` (≥ 0.7). No proprietary CUDA kernel or non-public dataset is needed to reproduce the runs.

New assets. The Given-49k dataset (49,151 puzzles, of which 48,138 are 17-clue) is the only new dataset introduced. We describe its construction in Appendix E; we will distribute it as a single JSONL file, along with the construction script, under the Creative Commons Attribution 4.0 (CC-BY-4.0) license. The Sudoku puzzles in Given-49k contain no personally identifiable information.

Existing assets. Table 27 lists every existing dataset and code base used in this paper. We use each asset as-is for evaluation or to retrain the corresponding baseline; we modified no asset beyond the documented filtering in Appendix E.

Table 27: Existing assets used in this paper. "Unknown" indicates that the original release does not specify a license; we use those assets only for the documented research purpose and follow the practice in the source repositories.

Asset	Source	Usage in this paper	License
RRN-180k Sudoku [9]	Palm et al., 2018 (public)	Train + eval	Unknown
Kaggle Sudoku (3M) [18]	Kaggle public dataset	Zero-shot eval only	CC0 (Public Domain)
RRN reference code [9]	Original GitHub release	Baseline retraining on RRN-180k and Given-49k	Unknown
Recurrent TR [10]	Yang et al., 2023 release	Baseline retraining on RRN-180k and Given-49k	MIT
Causal LM (Shah et al., 2024) reference code [3]	Shah et al., 2024 release	Baseline retraining on RRN-180k and Given-49k	Unknown
PyTorch Geometric (GAT) [25]	PyG library	Graph Coloring baseline	MIT
OR-Tools CP-SAT v9.15.6755	Google [19]	Non-learning Sudoku reference	Apache-2.0

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and the three contributions in Section 1 describe the integration of established mechanisms, controlled evidence on data efficiency, and evaluation across data regimes and relation templates. The claims are supported by Section 4 and the added studies in Appendices R–W. Comparisons distinguish learned inference from exact solving, separately trained domain configurations from zero-shot transfer, and single-seed diagnostics from multi-seed results.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 5 discusses hand-specified topology, the limited scope of single-seed corruption retraining, separately trained cross-domain models, failures to coordinate predictions on ambiguous puzzles, sequential recurrence, and the representation of higher-arity constraints. Automatic relation discovery and improved coordination among feasible completions remain open directions.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper makes no formal theorems or lemmas; its contributions are architectural and empirical. The equations in Sections 2 and 3 are definitional (CSP formulation, relational bias, hard axiom clamping, multi-step training objective) and do not require proofs.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 3 specifies the architecture in full, and Section 3.3 together with Appendix D covers the training objective and stabilization. Per-experiment hyperparameters, optimizer, learning-rate schedule, and data preprocessing are in Appendix O: Tables 14 and 15 give the standard configuration used by every main result and most ablations, and Table 17 gives the lightweight configuration used only by the Constraint Projection ablation (Table 4) and the auxiliary soft-constraint-loss study (Table 12). Dataset construction, including the Given-49k pipeline, is in Appendix E, and cross-domain experiment settings are in Appendix L (Table 16 for Binary Puzzle and Graph Coloring data sizes).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The model implementation and Given-49k construction script were supplied in the submission supplementary archive (Appendix X); a public repository URL is not yet available. RRN and Kaggle are public datasets. Appendix W specifies the separate CP-SAT implementation, versions, inputs, and scoring.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix E documents the train/val/test splits and the Given-49k construction. Appendix O lists the optimizer, learning-rate schedule, batch size, regularization, mixed-precision setting, and per-experiment epoch counts: Table 14 for the standard configuration and Table 17 for the lightweight configuration used only by the Constraint Projection and auxiliary-loss studies. Inference settings are reported in the same appendix, and Table 18 gives the stored checkpoint-entry breakdown, including buffers.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Headline CSP-RSB results use three seeds, and the original structural ablations, data-fraction studies, and CP controls use two (Appendix P). Table 4 reports two-seed mean±population standard deviation. The matched-backbone ladder and corruption settings each use one seed; additional graph configurations are single-seed development runs except for the reused headline row. We identify these limitations and do not assign multi-seed uncertainty to the CP-SAT reference.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix Q gives neural hardware and the original training-compute accounting. Appendix V separates measured GPU inference from analytical Trial & Error estimates; Appendix W reports the separate CPU environment, time limit, timing scope, and full-run duration. The original GPU-hour total excludes additional controls whose aggregate training times are not recorded here.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses synthetic constraint-satisfaction puzzles (Sudoku, Binary Puzzles, Erdős–Rényi Graph Coloring), with no human subjects, no personal data, and no scraped content beyond the publicly released benchmarks cited. The authors have read and conform to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The work may have positive impact by reducing the data, parameter, and compute requirements of constraint reasoning on puzzle-style CSPs. Direct negative societal impacts are limited, but CSP-RSB-style relational biases should not be used as the sole decision layer in high-stakes settings, because human-specified topologies can encode or propagate bias from how the constraints are written down.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

Justification: The artefacts supplied with the original submission are a CSP-RSB constraint solver and the curated Given-49k Sudoku split. They have no plausible high-risk dual-use profile: no language generation, no image generation, and no surveillance applications, so dedicated misuse safeguards are not warranted.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Existing neural datasets and implementations are credited in Appendix X. OR-Tools CP-SAT is additionally cited and its Apache-2.0 license identified. CP-SAT input provenance distinguishes the Radcliffe Kaggle release from the strategy-filtered subset distributed by Shah et al.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The CSP-RSB codebase and Given-49k Sudoku split are documented in Appendix X, with the construction procedure in Appendix E. The original supplementary package contains the neural training and evaluation scripts and environment instructions; no public repository URL is currently available. The separate CP-SAT protocol, version, and evaluation settings are documented in Appendix W. The neural code is under MIT, and Given-49k is intended for distribution under CC-BY-4.0. The puzzles contain no personally identifiable information.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper involves no crowdsourcing and no research with human subjects; all data come from programmatically generated or publicly released constraint-satisfaction puzzles.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper involves no human subjects, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are not a component of the CSP-RSB architecture, training objective, or solver inference. LLM assistance was used for manuscript editing and for implementing and checking the standard OR-Tools CP-SAT reference evaluation. That evaluation used fixed inputs and settings, with independent per-instance checks and statistical verification; this assistance is disclosed rather than characterized as copy-editing alone.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.