TINY RECURSIVE MESSAGE PASSING

Anonymous authors
Paper under double-blind review

ABSTRACT

Message passing neural networks have shown remarkable performance in algorithmic and constraint reasoning, due to their ability to encode the structure of the task directly in the architecture. Recent trends have, however, shown that better results can be achieved using recursive transformers trained with deep supervision and truncated recursions, which make extremely deep loops stable to train, and equipped with test-time procedures such as adaptive halting and discrete deduction lattices. In this work, we apply the same procedures to message passing networks, and show that the two ingredients are complementary, and lead to parameter efficient models: we obtain above 99% test accuracy on Sudoku-Extreme with a model of 16k parameters, $50\times$ fewer than the previous best result that trains in about 10 minutes on an H100, or in 9 hours on a CPU. We further show that the cost of truncating gradients grows with the number of rounds information needs to cross the graph, making the choice of wiring a trade-off between robustness to truncation and transfer across problem sizes. We conclude by reporting results comparable to the state-of-the-art on Maze-unique, and zero-shot transfer to larger instances on N-Queens and Boolean satisfiability.

1 INTRODUCTION

Structured reasoning tasks consist of problems whose solution is computed by executing a specific set of instructions, or by searching over a known set of constraints. Such tasks are traditionally solved by symbolic methods, which are exact and scale to arbitrarily large instances, but require the solving procedure to be designed by hand. Neural networks instead learn this procedure directly from examples. To be competitive, however, they must exploit the structure of the problem and generalize to instances larger than those seen during training. Initially, message passing methods Gilmer et al. (2017); Scarselli et al. (2008) dominated the neural subfield of algorithmic reasoning, as they are able to wire the interaction patterns of the problem directly inside the structure of the model. This way, no capacity is spent learning which variables constrain which, and the symmetries of the problem hold by construction rather than being learned from data Xu et al. (2019). A second property is their ability to perform an arbitrary number of message passing steps, that allows them to use more computation for more complex instances Selsam et al. (2019). This is particularly important when it comes to generalizing to problems that are harder than the ones encountered during training. Learning, however, is performed using backpropagation through time (BPTT) on the unrolled computational graph, which is memory expensive and hence limits in practice how deep the recursion can go.

Recent work has attacked this bottleneck by iterating a recurrent transformer model many times while replacing full backpropagation through the trajectory with a cheaper gradient. Deep equilibrium models differentiate through the fixed point of the forward pass, giving a parameter update in $\mathcal{O}(1)$ memory Bai et al. (2019; 2020), while tiny recursive models use a coarser approximation, running the recursion detached and differentiating only its final segment Jolicoeur-Martineau (2025). Two further ingredients make such deep loops trainable: deep supervision, which keeps the gradient path short, and adaptive computation, which spends depth where it is needed Graves (2016); Wang et al. (2025). Together these have brought Sudoku-Extreme within reach of small models: the work introducing the benchmark reported 55% accuracy with 27M parameters Wang et al. (2025), while models with 800k parameters now approach a perfect score Davis et al. (2026); Sghaier et al. (2026); Huang et al. (2026). Whether these approximations carry over to message passing is not obvious: the gradient that detached recursion discards is exactly the one that travels along the edges, and two wirings of the same constraint may depend on it very differently. We find that they do carry over, at a cost set by the propagation distance of the graph.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

The transformer models that achieved these results do not encode the structure of the problem. Can the parameter count of a recursive model be reduced further by giving it the structure in the way message-passing architectures do, while keeping the inference and learning procedures that made looped transformers work? Can we use this modern machinery to design message passing architectures that generalize well to larger scale tasks at test time? Deep supervision, detached recursion, and adaptive halting all presuppose a computation that decomposes into resumable segments carrying an explicit state. Recursive GNNs also carry such a state: at every round, each node reads the messages sitting on its edges together with its own memory, which is a tensor that can be paused and resumed like any other. What is missing is not the state but the training procedure built around it, which is the goal of this work. Our contributions are briefly as follows:

- We adapt the training and inference machinery of tiny recursive models to recursive GNNs. While deep supervision and truncated recursion carry over directly, adaptive halting and the deduction lattice require reformulation, namely per-vertex halting logits pooled with a smooth minimum, and a lattice defined over residual formulas for SAT.
- We show that encoding the structure and using the recursive machinery are complementary. On Sudoku-Extreme, a 32k-parameter recursive lattice transformer, and a recursive GNN with the same graph trained with BPTT (RRN) are outperformed by our method, that combines the two and exceeds 99% with 16k parameters. We obtain comparable results on Maze-unique.
- We show that the cost of truncating gradients grows with the propagation distance of the graph: dense wirings are robust to truncation, while factor graphs, which enable size transfer, require longer gradient windows. The lattice mitigates this cost by reducing the progress required within each supervised segment.
- We show that this class of models generalizes zero-shot to larger instances: on N-Queens, we train on 10×10 boards and evaluate on boards up to 16×16; on Boolean satisfiability, we train on medium-size SR and 3-SAT formulas and evaluate on hard ones. Our models outperform the baselines on nearly all sizes and splits, with 16k parameters on N-Queens and 64k on SAT.

2 Related Works

Recurrent message passing. The original graph neural network Scarselli et al. (2008) was already recursive: a single transition function, shared across vertices and iterations, is iterated to the fixed point of a contraction. GGNNs Li et al. (2015) drop the contraction, unroll a GRU for a fixed number of steps, and train by backpropagation through time. Implicit GNNs Gu et al. (2020) return to the equilibrium view. Because a GNN is a distributed algorithm whose number of rounds must grow with the problem Loukas (2019), weight-tied networks can be run longer at test time to solve larger or harder instances Tang et al. (2020); Schwarzschild et al. (2021); Bansal et al. (2022). Belief propagation is the canonical recursive message-passing algorithm, extended to the neural case with separate updates for variables and factorsKuck et al. (2020); Satorras & Welling (2021); Zhang et al. (2020) In this work, we will extensively use graphs with this topology. This is a special case of relational and heterogeneous networks Schlichtkrull et al. (2018); Wang et al. (2019). Our modules also address neighbours individually, as in the port-numbering model Angluin (1980); Hella et al. (2012) and CPNGNN Sato et al. (2019), and fall back on Deep Sets parameter sharing Zaheer et al. (2017); Maron et al. (2018) when ports are interchangeable. The closest precedents are Recurrent Relational Networks Palm et al. (2018), which solve Sudoku with deep supervision, and NeuroSAT Selsam et al. (2019), which passes messages over the bipartite structure of SAT and runs longer at inference than in training. Both backpropagate through the full unrolled trajectory, which bounds their depth in practice.

Deep recursive computation. A complementary line of work has investigated how weight-shared models can be iterated to greater depths without backpropagating through the entire trajectory. Implicit and equilibrium GNNs allow message passing to approach arbitrary depth through fixed-point computation and implicit differentiation Yang et al. (2025); Georgiev et al. (2024), while recurrent GNNs have explored adaptive message passing and the allocation of additional computation at test time Sun et al. (2026). In parallel, recursive reasoning models have shown that much simpler approximations, based on detached recursion, deep supervision, and adaptive halting, can train extremely deep recurrent computations efficiently on combinatorial reasoning problems Jolicoeur-

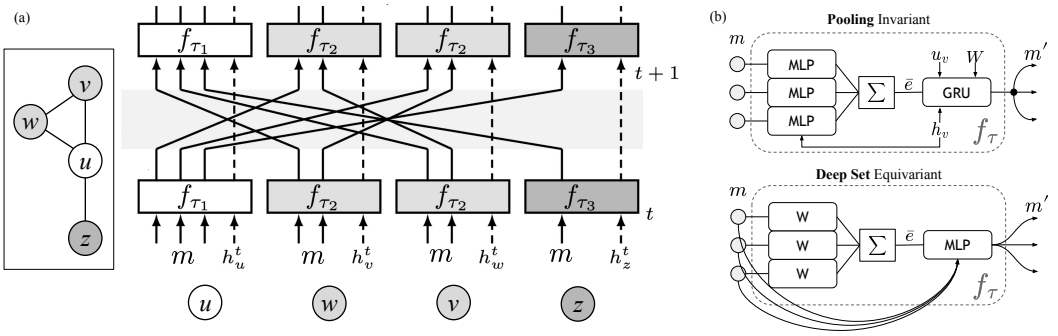


Figure 1: (a) On the left, a graph with four nodes (u, v, w, z), of types $(\tau_1, \tau_2, \tau_2, \tau_3)$ respectively. On the right, the trajectory of a single message passing step on the corresponding model. The three types correspond to three different neural modules f_τ, while modules of the same type are identical and share their weights. (b) Structure of the two main neural modules used in this work. The pooling module is invariant, and carries a latent state, used to compute the output when the iterative process has converged; the deep sets module is equivariant, and does not carry any latent memory.

Martineau (2025); Wang et al. (2025). In recent months, the performance of such recursive models on combinatorial tasks has further improved thanks to novel inference procedures, which either inject Gaussian noise in the search Huang et al. (2026); Sghaier et al. (2026), or project the output into a lattice to guide search Davis et al. (2026). In this work, we implement, adapt, and test all of them to see which one performs better and in which case.

3 Background

Let us assume we have a graph (V, E), where every vertex v holds a memory $h_v \in \mathbb{R}^m$, and every ordered pair of adjacent vertices carries a message, $m_{w\to v} \in \mathbb{R}^n$. In a message passing model, every round a shared function reads v's memory and the messages arriving at v, and writes a new memory together with one outgoing message per neighbour, which becomes the incoming message at that neighbour for the next round. In neural message passing, we assign every vertex a type $\tau(v)$, and all vertices of a given type share the same recurrent module f_τ, whose memory is defined as the latent state of a gated recurrent unit (GRU) Chung et al. (2014). For every vertex of a specific type τ, the module then performs the following operation:

$$f_\tau : \left(\prod_{w\in N(v)} \mathbb{R}^n\right) \times \mathbb{R}^{m_\tau} \times \mathbb{R}^{p_\tau} \longrightarrow \left(\prod_{w\in N(v)} \mathbb{R}^n\right) \times \mathbb{R}^{m_\tau}, \tag{1}$$

that maps messages, memory, and a fixed input u_v, to new outgoing messages and memory. For discrete inputs, u_v is a token embedding generated via a linear projection $u_v = Ex_v$. In practice, the way we define the modules for every vertex type is task dependent: we use larger ones for nodes that solve more complex tasks, and define their structures in a way that respects specific symmetries of the data. In this second case, the shape of f_τ depends on the invariances and equivariances its input/output messages respect. More generally, f_τ can give each edge a port or relation type recovers the typed, heterogeneous message passing discussed in the related works. The two modules below are the symmetric special cases of this view, in which all ports are treated as fully or partially interchangeable. Throughout, we will denote generic two-layer MLPs (a hidden linear layer, a ReLU, and an output linear layer) by MLP; we also drop all bias terms for simplicity. Two examples cover almost every vertex type in this paper:

Pooling module. This module is invariant to input permutations, and is hence used when the neighbours of v are interchangeable and v needs only one, shared answer. Every incoming message is encoded jointly with the memory by enc, averaged across neighbours, and passed through a recurrent update. This way, reordering $N(v)$ changes neither h'_v nor any outgoing message. The result is

broadcast back unchanged to every neighbour:

$$e_w = \mathrm{MLP}([h_v, m_{w\to v}]),$$
$$h'_v = \mathrm{GRU}([\mathrm{mean}_w(e_w), u_v], h_v),$$
$$m'_{v\to w} = W h'_v \;\; \forall\, w \in N(v).$$

Deep Sets module. This module is equivariant to input permutations, meaning that reordering the neighbours reorders the outputs correspondingly. It is used when the neighbours remain interchangeable, but v must answer each one individually. To achieve this, each message is embedded and pooled into a global summary, such as sum or a mean, which a MLP decodes the summary back by concatenating it with the initial message of the neighbour Zaheer et al. (2017):

$$\phi_w = \mathrm{ReLU}(W_\phi\, m_{w\to v}),$$
$$m'_{v\to w} = \mathrm{MLP}([m_{w\to v}, \mathrm{mean}_w(\phi_w)]) \;\; \forall\, w \in N(v).$$

During inference, first we inject the input via u_v to the model; then, the network is unrolled for a fixed number of rounds T, after which we compute the output via a linear projection from the latent states of the GRUs at convergence. During training, the output is used to compute a loss function, which in most cases is standard cross entropy. The gradients used to update the parameters of the neural modules are computed via BPTT across all T rounds. This is the practice we depart from in what follows, replacing it with deep supervision, truncated recursion, and adaptive halting so the same modules can be run without the linear memory cost of BPTT.

4 METHODS

We now discuss techniques already present in the literature that we use to scale up recursive models to complex reasoning tasks, and discuss how we have adapted them to work for message passing models: (1) the division of training and inference in three different loops, the *inner loop*, *outer loop*, and *deep supervision loop* Wang et al. (2025); (2) the use of truncated gradients, that only store a small number of iterations of the outer loop in memory Jolicoeur-Martineau (2025); (3) an adaptive computation time (ACT) head, that is, a small linear layer that learns whether the model has converged or not, and is able to stop the nested cycles before their fixed time Graves (2016); or, in alternative (4) the use of a discrete lattice that represents the intermediate inferred solution across steps Davis et al. (2026);

None of the techniques we use here alter the structure of the model presented in the previous section. The only difference is the addition of a new latent state, an *answer state* $a \in \mathbb{R}^{m_{\tau(v)}}$ that is updated before every outer loop and has been shown to be fundamental for propagating information across the different nested loops Wang et al. (2025). We have tested both the original formulation of the HRM, that uses an independent layer to perform the update of the answer state, and the TRM version, that uses the same module. Our early results showed that a smaller layer dedicated to the update of the answer state worked better, and hence we use this formulation throughout this work. To conclude, we have also tested the addition of noise on a trained model as done in probabilistic TRM, which allows one to compute multiple predictions from the same input by adding Gaussian noise at the end of every outer loop, and to pick the one that has the highest likelihood of being correct using the ACT head Sghaier et al. (2026).

Looped Iterations We divide computation into three nested recurrent loops. The *inner loop* consists of R rounds as defined in Eq. (1), during which messages x and vertex memories h evolve while the current answer state a, representing the current proposed solution of the network, is held fixed. At the end of the inner loop, the *outer loop* updates a and starts a new inner loop, so the complete recurrence can be written as

$$(x^{k+1}, h^{k+1}) = \Phi_\theta^R(x^k, h^k, u, a^k), \qquad a^{k+1} = \mathrm{LayerNorm}\big(\mathrm{GRU}(h^{k+1}, a^k)\big). \tag{2}$$

Here Φ_θ^R denotes R sequential applications of the shared vertex modules f_τ from Eq. (1), with u and a^k held fixed. In terms of size, the GRU cell for the outer loop is smaller than the main one, as it is always less than 10% of the total parameters of the model. Increasing either the inner- or outer-loop depth therefore increases computation while leaving the model parameters and state dimensions unchanged. At the end of one full sweep of inner and outer loops, the answer vector of every cell is

projected to the output space, and a parameter update is performed. Instead of moving to the next batch, however, this whole process is repeated multiple times in a row, thus forming our third level of nested loop, called *deep supervision*.

Truncated gradients. Due to the large number of nested loops, backpropagating through the entire trajectory would require storing all intermediate activations. While feasible on small models, this is problematic when scaling up to larger tasks. We therefore truncate the computational graph: earlier iterations are executed normally in the forward pass but their resulting state is detached before continuing, so gradients propagate only through a subset of the trajectory. If K outer iterations are executed and gradients are retained only for the last T, the computation is

$$s_K = G_\theta^T\big(\mathrm{sg}\big(G_\theta^{K-T}(s_0)\big)\big)\,, \tag{3}$$

where $G_\theta = U_\theta \circ F_\theta^R$ and sg denotes stop-gradient. Truncation therefore changes neither the forward dynamics nor the parameters or latent states of the model; it only limits temporal credit assignment and reduces training memory from the full recursive trajectory to the subset for which gradients are stored.

Adaptive computation time. To avoid always running the full supervision loop, we follow the previous literature and train a halting head that stops the training of a specific datapoint and moves to the following one Jolicoeur-Martineau (2025). Unlike previous work, however, a linear head defined on the vertices of a graph produces one logit per vertex via $q_v = W_h\ \mathrm{sg}(a_v)$, and not one for the overall latent state. This creates the need to aggregate the results of the individual logits, in order to reach a global, graph-level decision. As in all the problems considered in this work, a task is correct if and only if all of the individual cells are correctly classified, we pool with a smooth minimum $q = -\log \sum_v \exp(-q_v)$, so a single unconvinced vertex is enough to keep the whole graph from halting. To train such a module, we minimize a binary cross-entropy loss defined on every q_v, against whether the decoded value of its vertex v is already correct.

Interestingly, such a halting head can also be used to compute the confidence of the model with respect to a specific solution. To this end, let us assume we have two possible predictions for a given tasks, we could use the ACT head to select the most promising one. Following the probabilistic formulation of the TRM Sghaier et al. (2026), we implement a new way of scoring a test point: we run inference on a batch of K identical instances of the same datapoint, and we perturb the answer state with Gaussian noise at the end of every outer-loop update, resulting in K different predictions. Then, we use the halting head to select the solution with the highest confidence. In the original work, this has shown immense results, allowing a pre-trained TRM to go from 87% to 98% test accuracy Sghaier et al. (2026).

Discrete lattice. Deep-supervision steps are connected through a discrete lattice representing the partial solution accumulated so far Davis et al. (2026). For each variable, the lattice distinguishes values that have already been committed from those that remain unresolved; hence a state x_d represents not only a candidate solution, but also the information inferred up to deep-supervision step d. For example, in sudoku tasks, the row of each cell in x_d is a multi-hot subset of $\{1, \ldots, 9\}$: a blank cell starts with all nine candidates alive, and successive projections can only kill candidates, never revive them, until a cell is left with one alive candidate (committed) or, in a failure case, none; the support of x_d therefore encodes the full history of eliminations made up to step d, not merely a current best guess. After each inference phase, the model produces a distribution $p_\theta(\cdot \mid x_d)$ over possible values. A projection operator Π converts this continuous prediction into discrete deductions, committing sufficiently confident variables while retaining the previous value of all others:

$$[\Pi(p, x)]_i = \begin{cases} \arg\max_j p_{ij}, & \max_j p_{ij} \geq C, \\ x_i, & \text{otherwise}, \end{cases} \tag{4}$$

where C controls which predictions are committed. An intermediate board is invalid if there is a cell with every candidate solution being eliminated. The resulting lattice then becomes the problem presented to the next deep-supervision step,

$$x_{d+1} = \Pi\left(p_\theta(\cdot \mid x_d), x_d\right). \tag{5}$$

A partial solution that hits an internal conflict, that is, it fails the external validation, is reset and the process restarts from scratch, until a maximum number of deep supervision steps is completed. This

creates two kinds of failures: *timeout*, meaning that the model has failed to reach any solution after multiple trials, and *wrong*, meaning that the model produces an actual, wrong prediction. In short, this process can be seen as a way of guiding the recursive model towards the correct solution by providing each deep supervision loop the partial solution produced by the previous supervision step, rather than repeatedly solving the original instance from scratch.

5 EXPERIMENTS

Before applying message passing to a task, we must choose the graph that describes it, and hence the module each vertex runs, as tasks like Sudoku admit more than one sensible choice. A constraint problem consists of variables and constraints, each involving a few of the variables, and there are two natural ways to turn it into a graph: In a *factor graph*, every constraint is a vertex connected to the variables it involves, so variables communicate only through the constraints they share. In a *peer graph*, only variables are vertices, and two are adjacent whenever some constraint involves both, so every constraint becomes a clique. In a 4×4 Sudoku, the sixteen cells are constrained by twelve units: four rows, four columns, and four 2×2 blocks. The peer graph connects each cell to the seven cells it shares a unit with, for 56 edges. The factor graph connects each cell to its three units, for 28 vertices and 48 edges.

Peer or factor? The peer graph needs a single module, and two constrained cells exchange information in one round. The factor graph needs a second module, the Deep Sets module for the units, and two rounds per cell-to-cell exchange. On a 9×9 board, its cell-to-cell diameter is therefore 4 rather than 2. In exchange, a constraint over k variables costs k edges rather than $k(k-1)/2$, which gives 243 edges against 810 on a 9×9 board. More importantly, every unit sees all its cells at once and can reason about them jointly, for instance detecting that a digit fits in only one of its cells. A peer-graph cell pools its 20 peers together without knowing which unit each belongs to, which makes such reasoning much harder. Finally, the factor graph gives each cell a fixed number of ports regardless of problem size, which is what allows it to transfer across sizes (Sec. **??**). In short, the factor graph is more expressive and transfers across sizes, at the price of longer paths, which make it more sensitive to truncated gradients.

5.1 SUDOKU AND MAZE-UNIQUE

We test our method on the two constraint satisfaction puzzles most used in the recent literature: Sudoku-extreme Wang et al. (2025), boards of 9×9 cells to be completed, and Maze-unique Huang et al. (2026), a variant of Maze-Hard restricted to 30×30 mazes with a single correct path between an entrance and an exit cell. We will investigate different questions: which one of the two graph structures performs better and survives the truncation of the gradients. We also compare the performance of a lattice-based model against that of the noisy halting head. We train models of different sizes, wirings, and truncation regimes on both tasks. A factor graph model with 16k parameters trains on Sudoku-extreme in about ten minutes on a single H100, or in about nine hours on 32 CPU threads of an Intel Xeon Platinum 8480C. Training on Maze-unique takes longer, since every round of message passing now runs over a grid of 900 cells rather than a board of 81. To select which configurations of cycles and rounds to report, we use the ones that have reached the highest test accuracy over a single parametrization (average over 5 seeds).

Datasets. Sudoku-extreme has 1000 training, 300 validation, and 1000 test boards, selected as in Davis et al. (2026). The symmetry group of Sudoku is the direct product of two groups. The first is a geometric group, generated by permuting rows within a band, bands, columns within a stack, and stacks, and by transposition. The second is the group S_9 of digit relabelings. Our models are equivariant to the geometric group by construction, so we augment only with random digit relabelings. Maze-unique has disjoint train and test splits of 1000 mazes each; we double the training set by swapping entrance and exit. Its graph connects every pair of adjacent walkable cells.

Baselines. To separate the contributions of architecture and training, we use two kinds of baselines: The first is a 32k-parameter lattice deduction transformer Davis et al. (2026), which differs from our lattice models in its architecture; The second is a pair of RRNs Palm et al. (2018), one per wiring. They have the same shape and parameters as our models without lattice, but none of the machinery

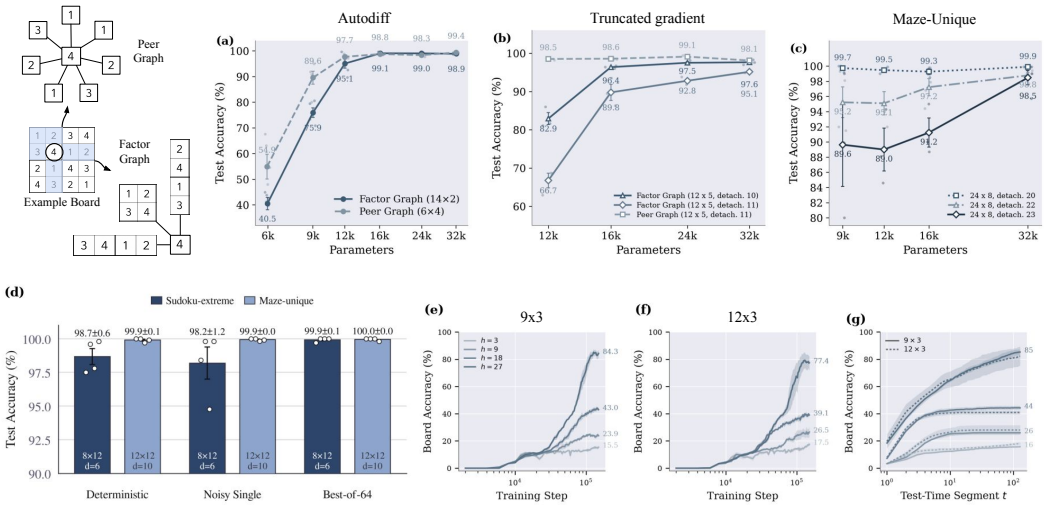


Figure 2: (a) Sudoku-extreme with the full gradient and the lattice; (b) The same comparison under truncated backpropagation; (c) Maze-unique, 24 cycles $\times$ 8 rounds, sweeping the truncation depth with the lattice; (d) Best models of each task without lattice, under the probabilistic ACT head, using models of size $32k$ and 1024 deep supervision steps at test time; (e) Training curves at 9 cycles and 3 rounds, when only h rounds carry the gradients (f) The same ladder at 12×3; (g) Test-time extrapolation: board accuracy as a function of the number of inference segments t for both segment lengths. Accuracy experiments are averaged over 5 random seeds.

the Methods section: they run 32 or 64 rounds and are trained with full BPTT, so they isolate the training procedure. We tuned the hyperparameters of all baselines for the small-parameter regime and removed dropout from the transformer. On Sudoku-extreme, the transformer reaches $71.60 \pm 6.13\%$, and the RRNs reach a maximum of $28.21 \pm 5.64\%$ (peer) and $32.21 \pm 4.11\%$ (factor). This shows that both the wiring of the graph that encodes the structure of the task in the model, and the training and evaluation regimes, matter for achieving good performance in the small parameter regime.

5.1.1 Results

With full backpropagation, both wirings exceed 99% on Sudoku-Extreme (Fig. 2(a)), while under truncated gradients the peer graph keeps its accuracy. The factor graph degrades and does not recover even when twice as many cycles carry gradients. This is consistent with path length: in the bipartite factor graph, two cells of the same unit are two rounds apart rather than one, so a fixed truncation window covers half as many cell-to-cell exchanges. Maze-unique, whose graph has a far larger diameter, amplifies the effect (Fig. 2(c)): it needs more cycles, rounds, and gradient-carrying steps to exceed 99%. The cost of truncating the gradient hence grows with the number of rounds information needs to cross the graph, and it is lowest for dense wirings such as the peer graph. The same holds for transformer models: as the attention graph is fully connected, detached recursion works better.

The accuracy results also hold when using the standard TRM recipe with the ACT head, and 32 deep supervision steps. We compare models with $32k$ parameters under one deterministic pass in Figure 2(d), the same model under one noisy pass, and the same model allowed several noisy passes chosen by the halting head. Dropping the lattice slightly affects the performance in the deterministic setting even when running 1024 deep supervision steps, but allows us to reach the best results once we use 64 noisy passes. The lattice and the noisy halting head address the same difficulty from opposite directions: the lattice commits to partial deductions explicitly and restarts on conflict, while the halting head perturbs the answer state and picks among the resulting candidates.

Truncated gradients. Figure 2(e) and (f) vary the number of gradient-carrying rounds h at fixed total depth. Accuracy rises sharply with h, so a short window is not enough to learn Sudoku-Extreme from scratch. Figure 2(g) shows this gap is further widened at inference: models trained with more gradients make better use of test-time iterations, while others tend to converge earlier. The credit

horizon hence sets a ceiling at training time, so it should be treated as a per-task hyperparameter that grows with the propagation distance and diameter of the graph. We also observe a stark difference in performance between (i) factor graphs and peer graphs, with the second being much more robust to the truncation of the gradients, due to its graph diameter being smaller, and (ii) between lattice and non-lattice methods: the lattice provides a simpler intermediate supervision signal, that allows the model to make improvements even using the small number of steps. This may not be the case for the standard supervision, as filling the next sudoku cell may need a larger number of backtracking steps than the ones that carry gradients, hence leaving the model with no useful signal. Further discussions and experiments can be found in the supplementary material.

Table 1: Zero-shot size transfer on N-Queens.

	Test board size						
Model	10×10	11×11	12×12	13×13	14×14	15×15	16×16
RRN, $T = 32$	0.80 ± 0.00	0.54 ± 0.02	0.36 ± 0.06	0.15 ± 0.05	0.02 ± 0.02	0.01 ± 0.01	0.00 ± 0.00
RRN, $T = 64$	0.95 ± 0.00	0.84 ± 0.01	0.68 ± 0.05	0.46 ± 0.15	0.22 ± 0.19	0.09 ± 0.12	0.00 ± 0.00
RRN, $T = 128$	0.84 ± 0.16	0.72 ± 0.17	0.62 ± 0.13	0.46 ± 0.11	0.38 ± 0.21	0.32 ± 0.19	0.26 ± 0.14
Ours, FG 8×12	0.96 ± 0.03	0.93 ± 0.08	0.98 ± 0.01	0.86 ± 0.12	0.55 ± 0.27	0.38 ± 0.31	0.28 ± 0.26
Ours, FG 10×12	0.97 ± 0.02	0.94 ± 0.07	0.88 ± 0.14	0.80 ± 0.15	0.63 ± 0.44	$\mathbf{0.67} \pm 0.44$	$\mathbf{0.63} \pm 0.34$
Ours, FG 12×12	0.99 ± 0.01	$\mathbf{0.99} \pm 0.00$	$\mathbf{0.98} \pm 0.02$	$\mathbf{0.97} \pm 0.04$	$\mathbf{0.79} \pm 0.18$	0.58 ± 0.37	0.51 ± 0.43

5.2 N-QUEENS

To test generalization to boards larger than those seen in training, we use the N-Queens dataset Baek et al. (2026). Given an $N \times N$ board with some queens already placed, the task is to complete it to N queens, no two sharing a row, column, or diagonal. For every complete 10×10 solution we sample 60 partial boards, 20 each with 1, 2, and 3 queens. Since our models are equivariant to the eight symmetries of the square, we split by orbit under this group, so that no test board is a rotation or reflection of a training board. This gives 13,442 training and 2,372 test boards. The 10×10 test split is used for model selection, as the important performance are that of larger boards $N \in \{11, \dots, 16\}$. There, we sample 400 solutions, keep 1, 2, or 3 queens, giving ≈ 750 test boards per size.

Model. Each cell is a node with a recurrent hidden state and an answer trace, and receives an embedding of its clue (queen or empty) at every step. We compare two graphs. In the peer graph, two cells are adjacent whenever they share a line. In the factor graph, every row, column, diagonal and anti-diagonal is a factor node. Every cell lies on exactly four lines, so it has exactly four ports for every N, while factors pool over their variable number of cells; no part of the model depends on N. As a baseline, we run an RRN on the same factor graph with the same widths, data, and target selection, trained as a single segment of $T \in \{32, 64, 128\}$ rounds with full BPTT and evaluated at depth T. All models have 16k parameters. Training follows Sudoku-extreme, without the lattice and with the ACT head only. Since a partial board may have several completions, the target is the consistent solution with the lowest current cross-entropy; sampling a random solution instead pushes predictions toward the average of the solutions. Clue queens are clamped at decoding, so a board counts as solved if and only if it contains N queens with no two sharing a line.

Results. The factor graph transfers well beyond its training size. The 12×12 model solves at least 97% of boards up to 13×13 and 79% at 14×14. On 16×16 boards, our configurations solve 28–63% on average (Table 1). The peer graph solves 99% of 10×10 boards but none of the larger ones, since the number of neighbours of a cell grows with N. The RRN degrades faster than our models: $T = 64$, the strongest RRN in distribution, drops to 0% at 16×16, and only $T = 128$ retains 26%, while requiring more memory and training time than our models.

5.3 BOOLEAN SATISFIABILITY (SAT)

We finally consider SAT, a standard testbed for GNNs (Selsam et al., 2019; Ozolins et al., 2022; Chang & Liu, 2025). We focus on satisfying assignment prediction. A formula in conjunctive normal form (CNF) is a conjunction of clauses, each a disjunction of literals, where a literal is a Boolean

Table 2: Solve rate on G4SATBench, mean over three seeds. Zero-shot transfer models are trained on medium and evaluated on hard without further tuning. *Numbers taken from the original work.

	In-distribution				Zero-shot transfer	
Model	SR easy	3-SAT easy	SR medium	3-SAT medium	SR hard	3-SAT hard
NeuroSAT (Selsam et al. (2019))	79.79	80.59	37.25	41.61	5.19^*	11.34^*
GGNN (Li et al. (2015))	78.75	76.42	21.18	25.68	–	–
Ours, det.6, ACT ($k = 1$)	$79.1_{\pm 1.2}$	$75.6_{\pm 0.5}$	$36.6_{\pm 0.9}$	$42.8_{\pm 2.8}$	$8.0_{\pm 0.3}$	$17.7_{\pm 2.7}$
Ours, det.6, ACT best-of-8	$86.1_{\pm 0.8}$	$82.7_{\pm 0.6}$	$47.6_{\pm 2.8}$	$54.9_{\pm 2.5}$	$16.4_{\pm 1.1}$	$32.2_{\pm 3.0}$
Ours, det.6, ACT best-of-64	$89.6_{\pm 0.9}$	$86.7_{\pm 0.8}$	$55.8_{\pm 4.1}$	$63.2_{\pm 1.8}$	$\mathbf{26.7}_{\pm 3.1}$	$\mathbf{44.8}_{\pm 2.8}$
Ours, lattice, det.10, 8×40	$\mathbf{99.7}_{\pm 0.1}$	$\mathbf{99.5}_{\pm 0.2}$	$\mathbf{72.5}_{\pm 1.1}$	$\mathbf{76.1}_{\pm 1.3}$	$12.3_{\pm 1.1}$	$29.5_{\pm 3.8}$

variable or its negation. Given a satisfiable CNF formula, the model must assign a truth value to every variable so that all clauses hold.

Model. We use the VCG* graph of G4SATBench: a bipartite graph between variables and clauses, whose edges are typed by literal polarity. This is the factor graph structure of Sudoku and N-Queens, with variables as cells and clauses as factors. The two polarities become two port types, encoded by a learned embedding added to each message. Variables run the pooling module with a recurrent hidden state and an answer trace, from which one logit per variable is read out; clauses run the Deep Sets module. No parameter depends on the number of variables or on clause length, so the medium-trained model runs on hard formulas unchanged. As baselines we use NeuroSAT and GGNN trained with the unsupervised loss, as reported in G4SATBench. Our model has 64k parameters, $6.5\times$ fewer than NeuroSAT (413k) and $5\times$ fewer than GGNN (330k).

Setup. Since a formula admits many satisfying assignments, we minimize the unsupervised loss of G4SATBench, i.e. the negative log-probability that each clause is satisfied. Without the lattice, the ACT head scores clauses rather than variables and pools them with a smooth minimum corrected for the number of clauses. With the lattice, each variable is undecided, committed, or in conflict, and the loss is computed on the residual formula left by the committed variables. At inference, instead of the projection of Eq. (4), we pick a random unsatisfied clause, commit one of its variables so that its literal becomes true, and restart on conflict. Conflicts are checked exactly, so the model never returns a wrong assignment, only timeouts.

Results. Table 2 reports the results. With a single rollout, our model roughly matches NeuroSAT in distribution (within 1.2 points on three of four splits and 5 points below on 3-SAT easy) and outperforms GGNN on the medium splits. Under zero-shot transfer to hard instances, the same single-rollout model already exceeds NeuroSAT by a factor of 1.5 (8.0 vs. 5.2 on SR, 17.7 vs. 11.3 on 3-SAT). Selecting among noisy rollouts with the halting head improves the solve rate further, and more so out of distribution. Best-of-64 gains about 20 points on the medium splits and reaches 4–5$\times$ the solve rate of NeuroSAT on hard instances. The lattice model is the strongest in distribution: it solves over 99% of easy formulas and nearly doubles NeuroSAT on medium ones. On hard instances, however, it falls behind best-of-8. Best-of-k and lattice search both spend more test-time computation than the single-pass baselines.

6 Conclusion

This work shows that the machinery developed for looped transformers (deep supervision, detached recursion, and the discrete deduction lattice) is not tied to transformers. Applied to typed message passing, it complements the structure encoded in the graph: neither ingredient alone solves Sudoku-Extreme in the small-parameter regime, while together they allow models with 16k and 32k parameters to match much larger ones. The machinery does not transfer at no cost, however: the price of truncating gradients grows with the propagation distance of the graph, making the choice of wiring a trade-off between robustness to truncation and transfer across sizes. Finally, the parameter savings come from giving the model the structure of the problem rather than spending capacity to learn it, which is both the point and the caveat of the comparison.

Reproducibility Statement

We provide a comprehensive description of the model and the experimental setup in the main paper; we also provide appendices for all training details. We will release the code in case of acceptance.

Ethics Statement

This work tackles fundamental machine learning problems. There are no specific ethical concerns to flag, besides that of fundamental progress in the field of AI and machine learning.

AI Usage

We have used LLMs for checking and polishing writing, for help with coding all the experiments in the paper. We reviewed all AI-assisted outputs and took steps to verify their correctness and reliability.

References

Dana Angluin. Local and global properties in networks of processors. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pp. 82–93, 1980.

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

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. Curran Associates, Inc., 2019.

Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Multiscale deep equilibrium models. *Advances in neural information processing systems*, 33:5238–5250, 2020.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. *Advances in Neural Information Processing Systems*, 35:20232–20242, 2022.

Wenjing Chang and Wenlong Liu. SAT-GATv2: A Dynamic Attention-Based Graph Neural Network for Solving Boolean Satisfiability Problem. *Electronics*, pp. 423, 2025.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Liam Davis, Leopold Haller, Alberto Alfarano, and Mark Santolucito. Lattice deduction transformers. *arXiv preprint arXiv:2605.08605*, 2026.

Dobrik Georgiev, JJ Wilson, Davide Buffelli, and Pietro Liò. Deep equilibrium algorithmic reasoning. *Advances in Neural Information Processing Systems*, 37:33638–33667, 2024.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. Pmlr, 2017.

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

Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. *Advances in neural information processing systems*, 33:11984–11995, 2020.

Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. In *Proceedings of the 2012 ACM Symposium on Principles of distributed computing*, pp. 185–194, 2012.

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

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

Jonathan Kuck, Shuvam Chakraborty, Hao Tang, Rachel Luo, Jiaming Song, Ashish Sabharwal, and Stefano Ermon. Belief propagation neural networks. *Advances in Neural Information Processing Systems*, 33:667–678, 2020.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural SAT solver. In *IJCNN*, 2022.

Rasmus Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. *Advances in neural information processing systems*, 31, 2018.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems. *Advances in neural information processing systems*, 32, 2019.

Victor Garcia Satorras and Max Welling. Neural enhanced belief propagation on factor graphs. In *International Conference on Artificial Intelligence and Statistics*, pp. 685–693. PMLR, 2021.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.

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*, 2019.

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

Junshu Sun, Wanxing Chang, Qingming Huang, and Shuhui Wang. Adaptive recurrent message passing for test time computing on graphs. *arXiv preprint arXiv:2606.22462*, 2026.

Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. *Advances in Neural Information Processing Systems*, 33:15811–15822, 2020.

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

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The world wide web conference*, pp. 2022–2032, 2019.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019.

Yongyi Yang, Tang Liu, Yangkun Wang, Zengfeng Huang, and David Wipf. Implicit vs unfolded graph neural networks. *Journal of Machine Learning Research*, 26(82):1–46, 2025.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Zhen Zhang, Fan Wu, and Wee Sun Lee. Factor graph neural networks. *Advances in Neural Information Processing Systems*, 33:8577–8587, 2020.

A TRUNCATED CREDIT ASSIGNMENT ON SUDOKU-EXTREME

We study how the number of rounds that carry gradients, h, affects learning. A model with C cycles of R rounds runs $C \cdot R$ message-passing rounds per deep-supervision step. Gradients flow only through the last h of these rounds, and all earlier rounds are detached. We consider two regimes: 32k-parameter models without lattice, trained with the probabilistic ACT head (Fig. 3(a, b)), and 16k-parameter lattice models (Fig. 3(c, d)).

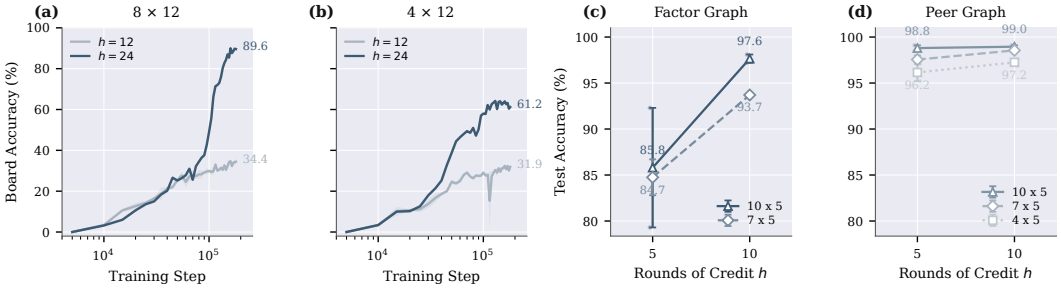


Figure 3: Truncated credit assignment on Sudoku-Extreme. Gradients flow only through the last h rounds of each deep-supervision step. (a, b) Validation board accuracy during training of the 32k-parameter model without lattice (probabilistic ACT head, at most 32 deep-supervision steps), for $C \times R = 8 \times 12$ and 4×12 and $h \in \{12, 24\}$. Shaded bands show $\pm$ SE over seeds. (c, d) Test accuracy on 1000 boards of 16k-parameter lattice models after 20k training steps, as a function of h, for the factor graph (10×5, 7×5) and the peer graph (10×5, 7×5, 4×5). Markers show the mean over 2 seeds, error bars $\pm$ SE, and dots individual seeds.

The gradient window sets the ceiling. In Fig. 3(a), the two curves share the same forward computation of 96 rounds and differ only in how many of them carry gradients. With $h = 24$, the model reaches 89.6%; with $h = 12$, it stalls at 34.4%. The 4×12 model in (b) shows the same pattern, rising from 31.9% to 61.2%. In both panels, the two curves overlap for roughly the first $5 \cdot 10^4$ steps before the $h = 12$ runs flatten out. A short window is therefore enough to learn the first, easy deductions, but not to keep improving beyond them. We conjecture that these later deductions require chains of inference longer than the window.

Forward depth and gradient depth interact. At $h = 12$, doubling the forward pass from 48 to 96 rounds changes the final accuracy by only 2.5 points: extra detached rounds add little when too few rounds are trained. At $h = 24$, the deeper model is far better (89.6% vs. 61.2%), so additional forward rounds do pay off once the window is long enough. This last comparison is confounded by the learning rate, however, since the 8×12, $h = 24$ runs use $7.5 \cdot 10^{-4}$ instead of $1.5 \cdot 10^{-3}$.

Dense wirings tolerate short windows. The lattice models in Fig. 3(c, d) show that how much truncation hurts depends on the wiring. For the factor graph, halving the window from $h = 10$ to $h = 5$ costs 11.8 points at 10×5 and 9.0 points at 7×5. It also makes training unreliable: at 10×5 and $h = 5$, the two seeds end 13 points apart. For the peer graph, the same change costs at most 1.1 points for every shape, and every configuration stays above 96%, including 4×5 with $h = 5$. In this setting, the gradient flows through the final cycle only.

Graph distance explains the gap. The cell-to-cell diameter is 2 in the peer graph and 4 in the factor graph, so a five-round window crosses the peer graph more than twice but the factor graph only about once. Forward depth still matters mildly for both wirings: at fixed h, accuracy increases from 4×5

to 7×5 to 10×5, consistent with (a, b). Together, these results suggest that the gradient window should be sized relative to the distances information must travel in the graph, rather than relative to the total depth of the forward pass.

REPRODUCING THE SUDOKU EXPERIMENTS

This section gives everything needed to reproduce the 16k-parameter factor-graph result on Sudoku: the architecture (§S1), the clique ablation (§S2), the exact training and evaluation recipe of the best recorded run (§S3), and a reference table of hyperparameters (§S4).

S1. FACTOR-GRAPH ARCHITECTURE

A puzzle is represented as a lattice $x \in \{0,1\}^{81\times 9}$ of multi-hot candidate sets, one per cell: a given cell is one-hot and a blank cell is all-ones. The lattice is encoded once, by a linear map, into a clue embedding c. The model is a fixed factor graph over the 81 cells and the 27 constraint units of Sudoku (9 rows, 9 columns and 9 boxes, each of arity 9), with every cell belonging to exactly 3 units. It is built from exactly two learned modules: a single SITE, shared by all cells, and a single RELATION, shared by all units, rather than one module per row, column or box.

Cell module (SITE). Cell i holds a recurrent state s_i and receives one message $m_{i,k}$ on each of its three ports ($k = 1, 2, 3$). Each message is encoded jointly with the state, and the codes are mean-pooled across ports, which makes the update permutation-invariant. The pooled code, together with the clue c_i and the current answer y_i, drives a $\mathrm{GRUCell}$ followed by a $\mathrm{LayerNorm}$ that updates s_i. A linear *emit* map then produces a single belief b_i, broadcast unchanged to all three outgoing ports:

$$e_{i,k} = \mathrm{MLP}_{\mathrm{enc}}([s_i; m_{i,k}]), \quad \bar{e}_i = \tfrac{1}{3}\textstyle\sum_k e_{i,k}, \quad s'_i = \mathrm{LN}\big(\mathrm{GRU}([\bar{e}_i; c_i; y_i], s_i)\big), \quad b_i = W_{\mathrm{emit}} s'_i.$$

Unit module (RELATION). All 27 units share the same Deep Sets block, whatever their kind. Each member belief u_k ($k = 1, \ldots, 9$) is embedded by ϕ, and the embeddings are summed into a permutation-invariant summary z. The map ρ then combines z with each individual member, so that every member receives a reply of its own. The block is therefore permutation-equivariant over the members of a unit:

$$\phi(u_k) = \mathrm{ReLU}(W_\phi u_k + b_\phi), \qquad z = \textstyle\sum_k \phi(u_k), \qquad r_k = \rho([u_k; z]).$$

Rounds, cycles and readout. One *round* consists of a synchronous firing of every SITE and every RELATION, together with two routing gathers. The cell-to-unit gather places b_i on the ports of the three units containing cell i, and the unit-to-cell gather returns each r_k to the port of the corresponding cell. A *cycle* consists of R rounds, after which a separate REFRESH module, $y \leftarrow \mathrm{LN}(\mathrm{GRU}(s, y))$, updates the answer trace y. A supervision step unrolls N cycles. Three linear heads (keep/kill, symbol and board conflict) are read off y after every cycle, which provides deep supervision.

Decoding. Decoding is a DPLL-style projection. A candidate is removed when its keep probability falls below the threshold θ. The board is declared solved when every cell is a singleton and no conflict is detected. Otherwise, one multi-valued cell is sampled and pinned to a symbol drawn from the tempered softmax of its symbol head. The projection is interleaved with the recursion, and no separate search is run inside a forward pass.

S2. CLIQUE VARIANT

The clique variant is a structural ablation of the factor graph that removes the unit level entirely. The RELATION module is deleted, and each of the 27 units is expanded into a pairwise clique: two cells are *peers* whenever they share a row, a column or a box. Each cell therefore has one port per peer instead of one per unit, which gives 20 ports instead of 3 (8 row-mates, 8 column-mates, and the 4 box-mates outside that row and column). Each peer reads the belief b_i directly from cell i, so a round is a single cell-to-cell hop instead of the cell→unit→cell exchange of the factor graph. As in §S1, R rounds make up one cycle.

The parameters freed by removing RELATION are reallocated to a wider SITE (a larger `hidden`). A clique and a factor graph at the same nominal budget (e.g. 16,485 versus 16,543 parameters at the

16k scale) therefore differ in *how* the budget is spent, not in its size, and comparing them compares topologies rather than model sizes.

S3. Training and Evaluation of the Best 16k Model

The reported 16k result uses the factor graph of §S1 with $R{=}2$ rounds per cycle and $N{=}14$ cycles per step. It is trained with the full gradient (no detached leading cycles) and deep supervision on all 14 readouts.

Persistent pool. Training maintains a pool of 512 puzzle states that persist across optimizer steps. At each step, the whole pool is passed once through the unrolled recursion to compute the loss (a weighted BCE on keep/kill, a symbol cross-entropy and a board-level conflict BCE). It is then passed once through the threshold projection, without gradient, to detect solved or conflicting entries. An entry is replaced by a fresh draw from the data stream in three cases: when it is solved and verified against the ground truth, when a detected conflict is confirmed, or when it reaches an age of 100 steps. Each fresh draw applies a random relabeling of the 9 symbols and one of three start kinds: the raw puzzle, the puzzle correctly pre-filled on a random fraction of its blanks, or the same pre-filling with some cells corrupted. The model thus learns both to solve a board from scratch and to repair an inconsistent one.

Optimization. We use AdamW with weight decay on matrix parameters only, a linear warmup followed by cosine decay, gradient-norm clipping at 1.0, bfloat16 mixed precision, and an exponential moving average (EMA) of the weights with Adam-style bias correction. The EMA weights are the ones checkpointed and evaluated. All sources of randomness (`torch`, CUDA and NumPy) are seeded once before the model is constructed, so a run is reproducible given its seed. The best recorded run (seed 5) solves 996/1000 test puzzles under the protocol below.

Model selection and test protocol. Starting at step 12,000, a checkpoint is scored every 2,000 steps on 500 validation puzzles, with a budget of 300 rounds and accept/reset checks every 4 rounds. The best-validating checkpoint is scored once on 1,000 held-out test puzzles, with a budget of 1,000 rounds and a check at every round. Each puzzle is attacked by 64 parallel stochastic chains, each of which resets to the initial puzzle whenever a conflict is detected. The first chain that the model itself flags as solved is accepted.

S4. Hyperparameters

Table 3: Hyperparameters of the 16k-parameter factor-graph recipe. Widths follow §S1: `dim` is the width of messages and of the clue embedding, `state` the width of a cell's recurrent state, `hidden` the inner width of both modules, and `y` the width of the answer trace.

Architecture	
Widths (`dim`, `state`, `hidden`, `y`)	$(9,\ 24,\ 48,\ 12)$, 16,543 parameters
Rounds per cycle R / cycles per step N	2 / 14
Detached leading cycles	0 (full gradient)
Optimization	
Optimizer steps	20,000
Pool size (= batch size)	512
Peak learning rate	6.8742×10^{-3}
Weight decay (matrix parameters only)	0.04821
Warmup fraction / schedule	0.1009, linear $\rightarrow$ cosine to a 5% floor
Adam betas	$(0.9,\ 0.95)$
Gradient clipping (norm)	1.0
EMA decay	0.9995
Pool max age	100 steps
Precision	bfloat16 autocast (GPU)
Seed (best recorded run)	5
Loss weights	
Keep/kill BCE, positive / negative weight	4.0 / 0.5
Symbol cross-entropy weight	0.2
Board-conflict BCE weight	0.1
Projection and decoding	
Keep threshold θ	0.1
Board-conflict threshold θ_{cls} (train / eval)	0.5 / 0.8
Pinning softmax temperature	1.5
Evaluation protocol	
Milestones	every 2,000 steps from step 12,000
Validation: puzzles / round budget / check every	500 / 300 / 4
Test: puzzles / round budget / check every	1,000 / 1,000 / 1
Chains per puzzle	64