CAN A ONE-LAYER TRANSFORMER SOLVE HARD SUDOKU? TEST-TIME COMPUTE, GENERALIZATION, AND MECHANISTIC ANALYSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern AI agents pair a language model with a harness that executes actions, maintains state, and returns feedback. We study where the boundary between the two lies in its simplest form: a *one-layer, one-head Transformer* paired with a *simple harness*, on Sudoku. At each step, the Transformer observes the grid and the candidates of every cell and selects either a cell–digit placement or REVERT; the harness applies the action, propagates constraints, and keeps the search history needed to backtrack. We train the Transformer by imitating a minimum-remaining-values (MRV) solver on easy puzzles only. We show that the Transformer generalizes from easy to hard puzzles: applied repeatedly, it solves all held-out easy, medium, and hard puzzles and 99.8% of extreme puzzles, whose searches run to thousands of actions, and harder puzzles need only more test-time compute. Thanks to its simple structure, we can trace how the model computes each action: attention selects the cell with the fewest candidates, the MLP favors the teacher's digit at that cell, and a dead-end cell shifts the output to REVERT. With only 0.87M parameters, the Transformer solves hard Sudoku almost perfectly, while prior single-model solvers use 5–27M parameters.

1 INTRODUCTION

Modern AI agents such as Claude Code and Codex pair a language model with a harness: the model chooses the next action, and the harness executes it, maintains the working state and memory, and returns feedback (Anthropic, 2025; OpenAI, 2025; Yao et al., 2023; Yang et al., 2024a). Recent work on harness engineering argues that capability now comes from the model and the harness working together, and that much recent progress comes from improving the harness rather than the model weights (Lopopolo, 2026; Rajasekaran, 2026; Weng, 2026; Zhang & Khattab, 2026). This division of labor raises questions that end-to-end success does not answer. Which parts of the reasoning does the *harness supply*, and which does the *model learn*? And *how small can the model be* while its decision rule stays reliable on *problems harder*, and *searches longer*, than those it was trained on? We study this boundary in its simplest form: a tiny model and a simple harness on a fully specified task.

In this paper, we focus on Sudoku, where the rules, the state, the legal actions, and a strong classical solver can all be specified exactly, while dead ends and variable search lengths still make the problem demanding. Our harness is a deterministic environment that keeps the grid, the candidates of every cell, and the search history; it applies each action and propagates its constraints. We adopt a one-layer, one-head Transformer that observes the grid and candidates and selects one of 729 cell–digit placements or REVERT, which undoes the most recent choice. The Transformer keeps no memory between decisions: the environment supplies candidate tracking and exact backtracking, and the Transformer supplies the choices. We train it from scratch by imitating a minimum-remaining-values (MRV) solver on easy puzzles, with traces capped at 200 forward actions.

We show that the Transformer generalizes from easy to hard puzzles. Applied repeatedly, it solves all held-out easy, medium, and hard puzzles and 99.8% of extreme puzzles, whose searches run to thousands of actions. We find that harder puzzles need more test-time compute, in the form of more steps of the same rule: easy to hard puzzles are solved within $8{,}192$ steps and extreme puzzles within $49{,}152$. At every budget, the Transformer solves the same number of puzzles as the MRV teacher,

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.

except for a single extreme puzzle, so it reproduces the teacher's decisions rather than finding a different route to the solution. We also find that learning this rule depends sharply on data: across training-set sizes, a single doubling raises held-out action accuracy from 39–50% to 90–99%.

Thanks to its simple structure, we can trace how the model computes each action. We find that attention implements the MRV choice of cell. The query of the readout token is fixed, so attention ranks cells by a learned score that penalizes each candidate, suppresses filled cells, and breaks ties by position; it selects the teacher's cell in all but five of 1.6 million teacher decisions. The attention output carries the selected cell's position and candidates to the MLP, which combines them to favor the teacher's highest-priority digit at that cell. The same score detects dead ends: a dead-end cell carries no candidate penalty, so attention selects it, and the model shifts its output to REVERT. We support each of these steps with interventions on candidates and activations.

Our contributions are as follows. **(i)** We show that a *one-layer, one-head Transformer*, trained by imitation on easy puzzles, achieves *easy-to-hard generalization* on Sudoku when combined with a simple harness that maintains the search state, and we measure a sharp *data-size transition* in learning its rule (§4). **(ii)** We give a *mechanistic account* of the learned policy, tracing cell selection, digit choice, and backtracking to attention and MLP computations tested by interventions (§5). To our knowledge, this is the *first work* to show that a one-layer Transformer with only 0.87M parameters, paired with a simple harness, solves hard Sudoku puzzles almost perfectly; prior single-model solvers use recursive networks with 5–27M parameters (Wang et al., 2025a; Jolicoeur-Martineau, 2025).

2 RELATED WORK

Our work relates to five bodies of literature: (i) *recursive reasoning and looped Transformers* (e.g., Dehghani et al., 2019; Giannou et al., 2023; Wang et al., 2025a; Jolicoeur-Martineau, 2025), (ii) *reasoning with agent harnesses* (e.g., Yao et al., 2023; Yang et al., 2026), (iii) *learning to solve and to search by imitation* (e.g., Palm et al., 2018; Lehnert et al., 2024; Shah et al., 2024; Ross et al., 2011), (iv) *grokking and the transition to generalization* (e.g., Power et al., 2022; Varma et al., 2023), and (v) *mechanistic interpretability* (e.g., Elhage et al., 2021; Olsson et al., 2022; Nanda et al., 2023a; Altabaa et al., 2025). Appendix A discusses each body of literature in detail, identifies the most closely related works, and explains how our work differs from them.

3 PROBLEM FORMULATION AND MODEL

Sudoku. A Sudoku puzzle is a partially filled 9×9 grid. The goal is to fill the empty cells with digits 1–9 so that each row, column, and 3×3 box contains every digit exactly once. We classify puzzles as *easy*, *medium*, *hard*, or *extreme* by the number of backtracks a separate preprocessing solver needs to solve them. The categories are fixed before any training and do not depend on the teacher or the model introduced below. See Appendix B.1 for details of the Sudoku task.

Our approach. We solve Sudoku with a *one-layer Transformer* that interacts with an environment. At each step, the Transformer observes the current state and chooses an action, and the environment applies it and returns the updated state. It also stores the search history, so it can revert to an earlier state. We train the Transformer by imitation learning on teacher trajectories from *easy* Sudoku puzzles. The Transformer learns the rule for a single action. Applied repeatedly, the Transformer achieves *easy-to-hard generalization*: it solves much harder puzzles, including extreme ones.

3.1 STATE, ACTIONS, AND TRANSITIONS

We formulate Sudoku solving as an interaction between the Transformer and an environment: at each step, the Transformer selects an action from the current state, and the environment applies it.

State representation. We number rows and columns from 1 to 9 and index the cell in a given row and column by $i = 9(\text{row} - 1) + (\text{col} - 1) \in \{0, \dots, 80\}$. At decision step t, the model observes state $s^{(t)} = ({S_{\text{g}}}^{(t)}, {S_{\text{c}}}^{(t)})$, where ${S_{\text{g}}}^{(t)}$ records the *current grid* and ${S_{\text{c}}}^{(t)}$ tracks the *remaining digit choices*. Specifically, ${S_{\text{g}}}^{(t)} \in \{0, \dots, 9\}^{81}$ records the value of each cell, with 0 denoting an

empty cell. The candidate state $S_c^{(t)} \in \{0,1\}^{81\times 9}$ satisfies $S_c^{(t)}[i, d-1] = 1$ if and only if digit $d \in \{1,\dots,9\}$ is a candidate for cell i. A candidate is a digit still available to try, excluding digits ruled out by the current row, column, or box and by earlier failed attempts. It need not lead to a complete solution. We write $S_{c,i}^{(t)} = \{d : S_c^{(t)}[i, d-1] = 1\}$ for the candidate set of cell i and $|S_{c,i}^{(t)}|$ for its size. Filled cells have no candidates. An empty cell with $|S_{c,i}^{(t)}| = 1$ is a singleton; one with $|S_{c,i}^{(t)}| = 0$ is a *dead-end cell*, and a state with a dead-end cell is a *dead end* (Figure 1).

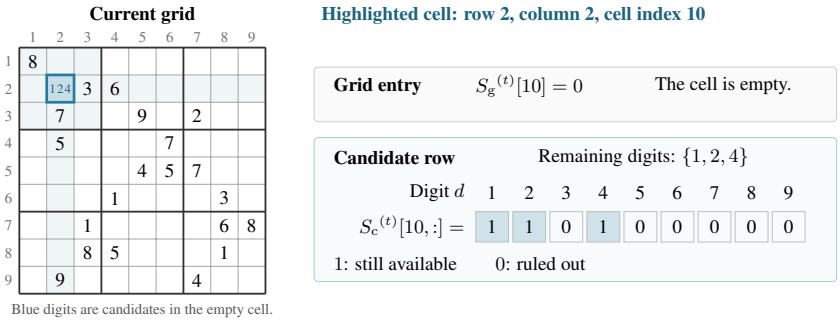


Figure 1: **A state example.** The highlighted empty cell has index $i = 10$, so $S_g^{(t)}[10] = 0$. Its row, column, and 3×3 box (light shading) rule out digits $3, 5, 6, 7, 8, 9$, leaving $\{1, 2, 4\}$. Thus the candidate row has ones at the entries for digits 1, 2, and 4, and zeros at the entries of all other digits.

Actions and transitions. At each step, the model can either try a digit in an empty cell (a forward action) or select REVERT. A forward action (i, d) is legal when the cell is empty, $S_g^{(t)}[i] = 0$, and the digit is still a candidate, $S_c^{(t)}[i, d-1] = 1$. We encode a forward action for cell i and digit d as $a = 9i + d - 1 \in \{0,\dots,728\}$ and reserve $a = 729$ for REVERT. The environment separately maintains a backtracking stack $U^{(t)}$, so the next state and stack are determined by $s^{(t)}$, $U^{(t)}$, and the selected action $a^{(t)}$. For a legal forward action or a REVERT action with a nonempty stack, the environment first updates an intermediate state $(\widetilde{S_g}, \widetilde{S_c})$ as follows:

- **Forward: try a choice and follow its consequences.** The environment pushes $(s^{(t)}, i, d)$ onto $U^{(t)}$ to obtain $U^{(t+1)}$. It copies the current state into $(\widetilde{S_g}, \widetilde{S_c})$, places digit d in cell i and clears that cell's candidates:
$$\widetilde{S_g}[i] \leftarrow d, \qquad \widetilde{S_c}[i, :] \leftarrow 0.$$
It then removes d from the candidates of other cells in the same row, column, or box.
- **Revert: undo a choice and rule it out.** The environment pops the most recent saved decision $(s^{(\tau)}, i, d)$ from $U^{(t)}$ to obtain $U^{(t+1)}$. It copies the saved state from before the choice into the intermediate state and removes digit d from cell i's candidates:
$$(\widetilde{S_g}, \widetilde{S_c}) \leftarrow (S_g^{(\tau)}, S_c^{(\tau)}), \quad \widetilde{S_c}[i, d-1] \quad \leftarrow 0.$$

The environment then applies constraint propagation:

$$s^{(t+1)} = (S_g^{(t+1)}, S_c^{(t+1)}) = \mathrm{Propagate}(\widetilde{S_g}, \widetilde{S_c}).$$

The operator repeatedly fills singleton cells and updates their peers' candidates until no singletons remain (Algorithm 2, Appendix D). Constraint propagation does not change the stack. An illegal forward action, or REVERT on an empty stack, leaves the state and stack unchanged.

3.2 ONE-LAYER TRANSFORMER

The model acting in this environment must map a state to one of the 730 action indices. We use the same one-layer, one-head Transformer at every decision step; it has 871,168 parameters. Given the current state $s^{(t)}$, the model outputs one logit for each of the 730 actions (Appendix Figure 7).

Embedding layer. We encode $s^{(t)}$ as a sequence of 83 tokens as follows:

$$\text{Input Sequence: } \big[\langle\text{BOS}\rangle, c_0^{(t)}, \ldots, c_{80}^{(t)}, \langle\text{EOS}\rangle\big], \qquad \text{Cell token: } c_i^{(t)} = \big(v_i^{(t)}, b_{i,1}^{(t)}, \ldots, b_{i,9}^{(t)}\big).$$

Cell i occupies token position $i+1$, and $\langle\text{BOS}\rangle$ and $\langle\text{EOS}\rangle$ occupy positions 0 and 82. Here $v_i^{(t)} = {S_{\text{g}}}^{(t)}[i] \in \{0,\ldots,9\}$ is the value of cell i, and $b_{i,d}^{(t)} = {S_{\text{c}}}^{(t)}[i, d-1] \in \{0,1\}$ indicates whether digit d is a candidate for cell i. The two special tokens use value codes 10 and 11, respectively. The model uses a value-embedding matrix $E_{\text{val}} \in \mathbb{R}^{12\times d_{\text{model}}}$ to map each cell value or special-token code to a learned vector. For each candidate digit d, it uses a candidate-embedding matrix $E_{\text{cand},d} \in \mathbb{R}^{2\times d_{\text{model}}}$; we write $E_{\text{val}}(\cdot)$ and $E_{\text{cand},d}(\cdot)$ for the row indexed by the argument. For cell i, the input embedding is

$$\mathbf{x}_i^{(t)} = \lambda \cdot \big(E_{\text{val}}\big(v_i^{(t)}\big) + \textstyle\sum_{d=1}^{9} E_{\text{cand},d}\big(b_{i,d}^{(t)}\big) + \mathbf{p}_i\big) \in \mathbb{R}^{d_{\text{model}}}, \qquad \lambda = \sqrt{d_{\text{model}}/2},$$

where $\mathbf{p}_i \in \mathbb{R}^{d_{\text{model}}}$ is a learned positional embedding, $d_{\text{model}} = 256$, and λ is a fixed scale applied to all input embeddings. Because the digit and each candidate bit have their own embedding, the input of a cell is a sum of per-feature vectors; Section 5 uses this additive structure to decompose the attention scores. We index $\mathbf{x}_i$ and $\mathbf{p}_i$ by the cell index i and use the subscript 82 for the $\langle\text{EOS}\rangle$ token. For $\langle\text{BOS}\rangle$ and $\langle\text{EOS}\rangle$, the value embeddings are $E_{\text{val}}(10)$ and $E_{\text{val}}(11)$, respectively, and their candidate embeddings are $\sum_{d=1}^{9} E_{\text{cand},d}(0)$.

Transformer block. Let $\mathbf{H}_0^{(t)} \in \mathbb{R}^{83\times d_{\text{model}}}$ contain the input embeddings in sequence order. The query, key, and value matrices are $\mathbf{Q}^{(t)} = \mathbf{H}_0^{(t)} W_Q^\top$, $\mathbf{K}^{(t)} = \mathbf{H}_0^{(t)} W_K^\top$, and $\mathbf{V}^{(t)} = \mathbf{H}_0^{(t)} W_V^\top$. The Transformer block applies self-attention followed by a SwiGLU MLP, each with a residual connection and followed by RMS normalization (post-norm):

$$\mathbf{H}_1^{(t)} = \text{RMSNorm}_\epsilon\big(\mathbf{H}_0^{(t)} + \text{softmax}\big(\mathbf{Q}^{(t)}\mathbf{K}^{(t)\top}/\sqrt{d_{\text{head}}}\big)\mathbf{V}^{(t)} W_O^\top\big),$$
$$\mathbf{H}_2^{(t)} = \text{RMSNorm}_\epsilon\big(\mathbf{H}_1^{(t)} + \big[\text{SiLU}\big(\mathbf{H}_1^{(t)} W_1^\top\big) \odot \big(\mathbf{H}_1^{(t)} W_2^\top\big)\big] W_3^\top\big).$$

Here $\mathbf{H}_1^{(t)} W_1^\top$ and $\mathbf{H}_1^{(t)} W_2^\top$ are the gate and up projections, respectively, $\text{SiLU}(x) = x/(1+\exp(-x))$, and $\odot$ denotes elementwise multiplication. The softmax is applied across sequence positions in each row, and $\text{RMSNorm}_\epsilon(\mathbf{y}) = \mathbf{y}/\sqrt{\|\mathbf{y}\|_2^2/d_{\text{model}} + \epsilon}$ is applied independently to each row, where $\epsilon = 10^{-5}$. We use $d_{\text{head}} = d_{\text{model}} = 256$ and $d_{\text{ff}} = 512$, with $W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{head}}\times d_{\text{model}}}$, $W_O \in \mathbb{R}^{d_{\text{model}}\times d_{\text{head}}}$, $W_1, W_2 \in \mathbb{R}^{d_{\text{ff}}\times d_{\text{model}}}$, and $W_3 \in \mathbb{R}^{d_{\text{model}}\times d_{\text{ff}}}$.

Action prediction. Let $\mathbf{h}_{82}^{(t)} = (\mathbf{H}_2^{(t)})_{82,:}^\top \in \mathbb{R}^{d_{\text{model}}}$ denote the final hidden state of the $\langle\text{EOS}\rangle$ token at position 82. The bias-free action head $W_{\text{act}} \in \mathbb{R}^{730\times d_{\text{model}}}$ produces

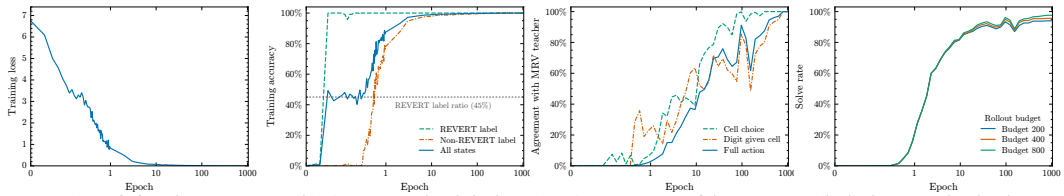
$$\mathbf{z}^{(t)} = W_{\text{act}}\mathbf{h}_{82}^{(t)} \in \mathbb{R}^{730}, \qquad a^{(t)} = \text{argmax}_{a\in\{0,\ldots,729\}}\ z_a^{(t)}.$$

In training and evaluation, we do not mask these logits using the candidates.

3.3 MRV TEACHER, ROLLOUTS, AND SUPERVISED LEARNING

MRV teacher. The environment specifies which actions are available but not which one to take. We obtain a target action for every state from a classical solver built on the minimum-remaining-values (MRV) heuristic (Haralick & Elliott, 1980): choose the empty cell with the fewest candidates, $|S_{\text{c},i}^{(t)}|$. When several empty cells tie for the fewest candidates, the teacher takes the one with the smallest cell index. The teacher then tries the cell's candidates in the priority order $1, 2, \ldots, 9$. If the state is a dead end, the teacher selects REVERT, which backtracks to an earlier decision. Both the tie-break and the digit order make the teacher deterministic, so every state has a single target action. Combined with the environment's propagation and backtracking, this teacher is a complete solver: MRV changes only the order of the search, not whether it reaches the solution (Appendix B.3).

Rollouts. Repeated interaction between a policy, either the teacher or the trained model, and the environment produces a rollout. A rollout starts from $s^{(0)}$, the puzzle after initial constraint propagation, with an empty stack. It ends when the grid is complete or when the rollout budget is exhausted, and it is *solved* if the completed grid matches the reference solution. The rollout budget counts every selected action, including REVERT and illegal forward actions.



(a) Training loss (b) Accuracy by label (c) Agreement with MRV (d) Solve rate by budget

Figure 2: Training dynamics. **(a)** Training loss and **(b)** next action accuracy on augmented training batches, with accuracy separated into REVERT-labeled and forward-labeled states. The gray line in panel (b) marks the 45% proportion of REVERT labels. **(c)** Rollout agreement with the teacher at a budget of 200 steps, separated into cell choice, digit given the cell, and full action. **(d)** Solve rate at budgets of 200, 400 and 800 steps. Panels (c) and (d) are both evaluated on the same 500 held-out easy puzzles.

Training data. We train the model to imitate the teacher. Before training, we generate teacher trajectories from $5{,}000$ puzzles in the easy training split using a forward-action cap of 200. Each forward state is paired with the action selected by the teacher, including actions on paths that later reach a dead end. For each REVERT action, we reconstruct the dead-end state before the environment processes the action and pair it with REVERT. The resulting dataset contains $158{,}215$ pairs, including $71{,}285$ REVERT targets (45.1%). At each training step, we independently *augment* every pair in the mini-batch: we apply a uniformly sampled digit permutation with probability $1/2$, a uniformly sampled rotation by a multiple of $90°$, and a horizontal or vertical reflection with probability $1/2$.

Training recipe. We train the model from scratch using cross-entropy loss. The loss is applied independently to state-action pairs, without backpropagating through environment transitions. We train for $1{,}000$ epochs with a batch size of 256 using AdamW (Appendix B.5).

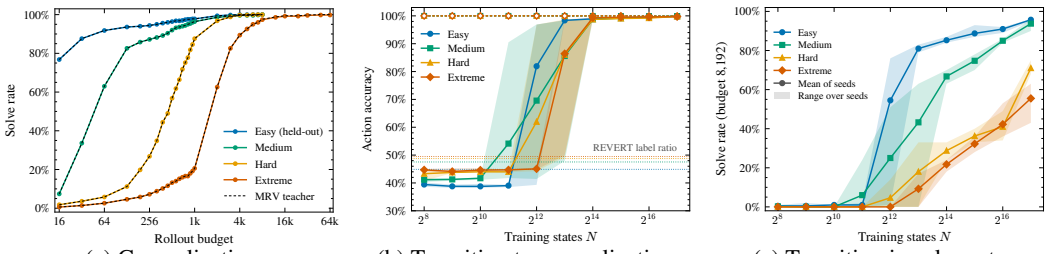
4 TRAINING AND GENERALIZATION

We use three metrics: (i) *Next action accuracy* is the fraction of states where the model selects the teacher's action. (ii) *Rollout agreement* is the fraction of steps in model rollouts at which the model's action matches the teacher's. (iii) *Solve rate* is the fraction of puzzles solved within the rollout budget.

Training dynamics. In Figure 2, the training loss decreases to nearly zero and next action accuracy reaches 100%. Early in training, the model predicts REVERT for every state, so overall accuracy equals the REVERT share of the training data (gray line in Figure 2b), and forward accuracy improves later. Appendix E separates missed and unnecessary backtracking, illegal placements, and legal teacher disagreements. At the plotted checkpoints from step $3{,}260$ onward, both backtracking errors are zero, while illegal placements dominate the remaining disagreement. As agreement improves, the solve rate increases, and the fully trained model attains the teacher's solve rate at all three budgets.

Generalization beyond the training distribution. We examine whether the model trained in Section 3.3 generalizes beyond its training distribution. On 500 held-out puzzles per difficulty (Figure 3a), the Transformer solves all easy, medium, and hard puzzles within $8{,}192$ steps and 99.8% of extreme puzzles within $49{,}152$ steps. This generalization is nontrivial: the model ($871{,}168$ parameters) selects among all 730 unmasked logits, so one illegal action can stall a greedy rollout, and although its training traces are capped at 200 forward actions, it selects thousands of consecutive valid actions on harder puzzles. Across the $2{,}000$ rollouts, it deviates from the teacher only once, due to an illegal digit. The MRV teacher, run on the same puzzles and budgets (dashed lines), solves the same number of puzzles as the model at every budget except this extreme rollout.

From memorization to generalization. To measure the transition as a function of dataset size, we train models on $N \in \{256, \ldots, 131072\}$ teacher states of each difficulty (Figure 3b,c). While all models attain near-100% training accuracy, small N yields near-zero solve rates and evaluation accuracy bounded by the REVERT prior, indicating pure memorization. As N crosses a difficulty-dependent threshold (later for hard and extreme puzzles), a single doubling of N raises evaluation accuracy from 39–50% to 90–99% in every seed, with large gains in solve rate (Figure 3b,c).



(a) Generalization (b) Transition to generalization (c) Transition in solve rate

Figure 3: Generalization beyond the training distribution. **(a)** Solve rate of the model trained on easy puzzles at different rollout budgets, evaluated on 500 held-out puzzles of each difficulty. **(b)** Next action accuracy on the training and evaluation data. **(c)** Solve rate at a budget of 8,192 steps on 100 held-out puzzles of each difficulty. Models in panels (b) and (c) are trained from scratch for 20,000 optimizer steps, without augmentation, on nested training sets of N states of a single difficulty; each point averages up to four seeds (Appendix B).

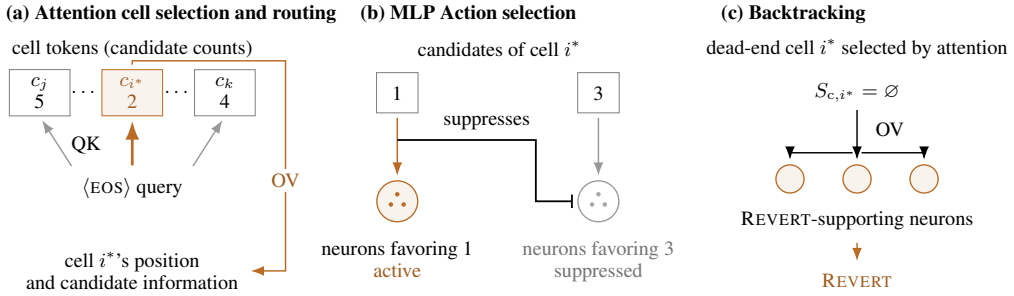


Figure 4: **Cell selection, action choice, and backtracking. (a)** The ⟨EOS⟩ query attends most strongly to cell i^*; OV carries its position and candidate information. Numbers below cell tokens give candidate counts. **(b)** For candidates $\{1, 3\}$ at cell i^*, digit 1 activates neurons favoring 1 and suppresses neurons favoring 3; the bar-ended connection denotes suppression. **(c)** An empty candidate set activates REVERT-supporting neurons. Orange marks the selected cell or favored response. Connections, arrow widths, and neuron populations are schematic.

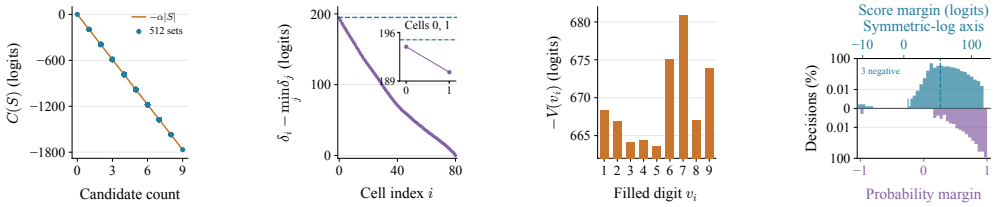
Takeaway. (i) *Training Progression*: Backtracking errors disappear before illegal placements. (ii) *Generalization in Hardness and Length*: Trained only on short, easy traces, the model solves nearly all held-out puzzles from easy to extreme. (iii) *Sharp Transition*: Generalization emerges within one doubling of the training set, at a difficulty-dependent threshold.

5 MECHANISTIC INTERPRETATION OF THE TRAINED MODEL

Summary. The trained model implements the teacher's rule through three linked mechanisms (Figure 4). *Attention routing* selects an empty cell with the fewest candidates and routes its position and candidates to the MLP (§5.1). *Action selection* uses MLP activity tied to the selected cell and preferred digit to produce the teacher's action (§5.2). *Backtracking* begins when attention selects a dead-end cell, which shifts support to REVERT (§5.3). We trace all three at the ⟨EOS⟩ token, whose hidden state feeds the action head; Appendix F gives the details.

5.1 ATTENTION: SELECT A CELL AND ROUTE ITS INFORMATION

Mechanism 1. Select and route cell information. The ⟨EOS⟩ token's attention is typically close to one-hot on an empty cell with the fewest candidates. Among tied cells, the one with the smallest index usually receives the most probability. The attention OV projection carries the attended cell's position and candidate information to the MLP.



(a) Contributions of 512 candidate sets (b) Positional contribution across 81 cells (c) Value advantage by filled digit (d) MRV selection margins

Figure 5: Attention logits favor minimum-candidate empty cells. **(a)** Contributions of all 512 candidate sets (points) and the linear prediction $-\alpha|S_{c,i}|$ (line), with $\alpha \approx 196.3$. Points overlap because deviations from the line are at most 0.682 logits. **(b)** Positional contributions, with the minimum advantage from one fewer candidate shown dashed. **(c)** Value-penalty magnitudes $-V(v_i)$ for digits 1–9, with a truncated y-axis. **(d)** Logit and attention-probability margins between the teacher-selected cell and its strongest competing cell.

Attention-score decomposition. To understand how the attention layer forms the one-hot attention distribution and selects the targeted cell, we analyze one decision step and omit the superscript (t). The input of the $\langle \text{EOS} \rangle$ token is identical across states, so its query $\mathbf{q}_{82} = W_Q \mathbf{x}_{82}$ is fixed in this single-block model. Define $\bar{\mathbf{p}} = \frac{1}{81} \sum_{j=0}^{80} \mathbf{p}_j$ and set $\mathbf{r} = \lambda W_K^\top \mathbf{q}_{82} / \sqrt{d_{\text{head}}}$, with λ from Section 3.2. Substituting the factored embedding into the attention score gives

$$\ell_{82,i} = \mathbf{q}_{82}^\top W_K \mathbf{x}_i / \sqrt{d_{\text{head}}} = b + V(v_i) + C_i + \delta_i. \tag{1}$$

The components in (1) are

$$b = \mathbf{r}^\top \big(E_{\text{val}}(0) + \textstyle\sum_{d=1}^{9} E_{\text{cand},d}(0) + \bar{\mathbf{p}}\big), \qquad V(v_i) = \mathbf{r}^\top \big(E_{\text{val}}(v_i) - E_{\text{val}}(0)\big),$$
$$C_i = \textstyle\sum_{d \in S_{c,i}} \kappa_d, \qquad \kappa_d = \mathbf{r}^\top \big(E_{\text{cand},d}(1) - E_{\text{cand},d}(0)\big), \qquad \delta_i = \mathbf{r}^\top \big(\mathbf{p}_i - \bar{\mathbf{p}}\big). \tag{2}$$

Here b is shared across cells, $V(v_i)$ is the value contribution relative to an empty cell, κ_d is the candidate contribution of digit d, C_i sums it over the candidates of cell i, and δ_i is the positional contribution of cell i relative to the mean over cells.

How candidate count and position determine cell selection. For empty cells, (1) reduces to $\ell_{82,i} = b + C_i + \delta_i$. Across all 512 candidate sets, $C_i \approx -\alpha|S_{c,i}|$, with $\alpha \approx 196.3$ and maximum deviation 0.682 logits, so each additional candidate lowers the score by about 196.3 at a fixed position. With the candidate set fixed, δ_i decreases monotonically with cell index, favoring earlier cells. The minimum gain from one fewer candidate exceeds the full positional range (194.952 versus 193.950 logits), so candidate count sets the primary ordering, while position usually resolves ties. Figure 5(a,b) shows these contributions; Appendix F.2.2 gives the exact bounds and their qualifications.

Why filled cells are usually not selected. Two-candidate cells account for 99.9% of forward decisions across a separate sample of 4,000 complete teacher trajectories (Table 2 in Appendix F.2.5), so we focus on them. Let j be an empty cell with $|S_{c,j}| = k$ candidates and i a filled cell. Since $V(0) = 0$ and $C_i = 0$, subtracting their scores cancels the shared baseline:

$$\ell_{82,j} - \ell_{82,i} = -V(v_i) + C_j + (\delta_j - \delta_i) \approx \underbrace{-V(v_i)}_{\text{value advantage}} - \underbrace{\alpha k}_{\text{candidate penalty}} + \underbrace{(\delta_j - \delta_i)}_{\text{positional difference}}. \tag{3}$$

The empty cell wins when its value advantage exceeds its candidate penalty plus any positional disadvantage. For $k = 2$, the measured worst-case bounds give $\ell_{82,j} - \ell_{82,i} \geq 663.5 - 393.3 - 194.0 > 76$ logits (Appendix F.2.3), so every two-candidate empty cell outranks every filled cell in the exact decomposition. The rarer three-candidate cases have a larger candidate penalty and can admit filled-cell attention winners; Appendix F.2 gives the bounds and exceptions.

Validating the attention mechanism. The attention argmax matches the teacher-selected cell with rare exceptions (Appendix F.2; margins in Figure 5d), and attention places nearly all probability on that cell, so we approximate the attention output by that cell's contribution.

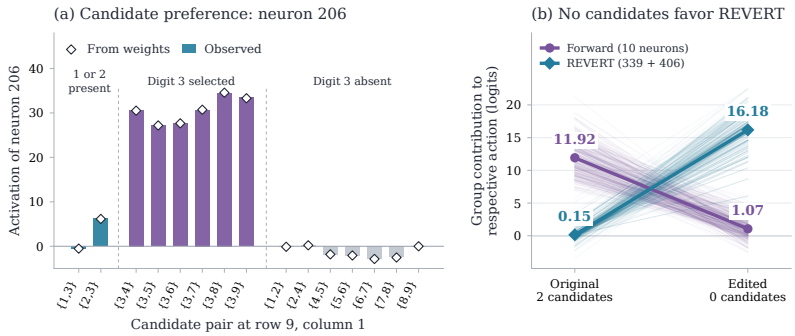


Figure 6: Candidate sets control digit selection and backtracking. **(a)** From left to right, neuron 206 has suppressed activity when digit 3 competes with a higher-priority candidate (blue), strong activity when digit 3 is selected (purple), and weak responses when digit 3 is absent (gray). Bars show observed means; diamonds show predictions from weights. Digits follow teacher-preference order. **(b)** Clearing the selected cell's candidates decreases the original action group's contribution and increases the contribution of two REVERT neurons. Faint lines show 324 paired states; bold lines show means. Appendix Figure 18 adds gate coefficients.

Attention OV projection. Let $W_{\mathrm{OV}} = W_O W_V$ denote the attention OV projection and define $\mathbf{\Delta}_d = \lambda W_{\mathrm{OV}}(E_{\mathrm{cand},d}(1) - E_{\mathrm{cand},d}(0))$. Following a similar additive decomposition to (2), linearity gives the exact OV contribution from cell i:

$$W_{\mathrm{OV}}\mathbf{x}_i = \mathbf{u}_0 + \mathbf{u}_{\mathrm{pos},i} + \mathbf{u}_{\mathrm{val}}(v_i) + \textstyle\sum_{d \in S_{\mathrm{c},i}} \mathbf{\Delta}_d, \tag{4}$$

where $\mathbf{u}_0$, $\mathbf{u}_{\mathrm{pos},i}$, and $\mathbf{u}_{\mathrm{val}}(v_i)$ are the shared, positional, and filled-value contributions, with $\mathbf{u}_{\mathrm{val}}(0) = 0$. Appendix F.3 gives the exact decomposition. Let $\mathbf{o}_{82}$ denote the attention output at the ⟨EOS⟩ token before residual addition. Here i^* denotes the attention argmax, which can differ from the teacher-selected cell. When attention concentrates on an empty cell i^*, $\mathbf{o}_{82} \approx W_{\mathrm{OV}}\mathbf{x}_{i^*}$. Position and candidate directions are nearly orthogonal (mean $|\cos| = 0.089$); the only near-collinear pair is digits 8 and 9, which the teacher's priority order never distinguishes (Appendix F.3.2).

5.2 MLP ACTION SELECTION: COMBINE CELL IDENTITY AND DIGIT PREFERENCE

Mechanism 2. Combine cell identity and digit preference to choose an action. The MLP plays the dominant role in choosing the cell–digit action. It uses position information to distinguish actions at different cells and candidate information to favor the teacher's preferred digit. Higher-priority candidates suppress neuron groups supporting lower-priority digits.

The MLP accounts for most of the action margin. To measure the contribution of the MLP to the final action, we define the action margin as the difference between the logit of the teacher action and the largest competing logit:

$$m_{\mathrm{action}} = z_{a^*} - \max_{a \neq a^*} z_a, \tag{5}$$

where a^* is the teacher action and the maximum ranges over the other 729 action indices. We decompose the margin into contributions from the residual and the MLP output. Across 600 sampled forward states from model rollouts, the MLP contributes 92.7% of the mean action margin. Setting its output to zero reduces agreement with the teacher from 100% to 30.8%. Thus, the MLP plays the dominant role in choosing the teacher's action on these states (Appendix F.4.1).

Activations track the selected cell and the teacher's preferred digit. We next test whether the observed MLP activity reflects this joint dependence on cell identity and digit preference. At a fixed cell, different candidate pairs produce similar activations when the teacher chooses the same digit. Changing either the preferred digit or the selected cell changes activations much more (Appendix Figure 14). Grouping the MLP activation vectors by the teacher's cell–digit action, the group means account for 90.04% of total variance. MLP activity is organized primarily by the selected cell and preferred digit, with smaller differences between candidate sets for the same action (Appendix F.4.2).

Higher-priority candidates suppress lower-priority choices. To illustrate how an MLP neuron implements the teacher's digit priority, consider neuron 206 at row 9, column 1. Its output projection most strongly supports digit 3 among the nine digit actions at this cell, so positive activation promotes placing digit 3 there. Figure 6(a) shows that its activity is strong only when digit 3 is the highest-priority candidate. Neuron groups show the same pattern (Appendix F.4.6): the preferred candidate raises a group's activation, and higher-priority candidates lower it. Thus, on average, the highest-priority candidate promotes its own neuron group and suppresses those of lower-priority candidates.

Position information distinguishes cell–digit actions. The position vector changes the MLP's response to a fixed candidate set, distinguishing actions at different cells (Appendices F.4.5, F.4.6).

5.3 Backtracking: attend to a dead-end cell and select Revert

> **Mechanism 3. Select Revert at a dead end.** Minimum-candidate attention selects a dead-end cell, an empty cell with no candidates. The attention OV projection carries this cell's information to the MLP, reducing forward-action support and increasing support for Revert. The environment then restores an earlier state and continues the search.

Attention selects a dead-end cell. At a dead end, at least one empty cell has no remaining candidates, a dead-end cell (Section 3.1). For this, $V(0) = C_i = 0$, so its attention score is $b + \delta_i$ in (1). The candidate-count ordering puts it above every empty cell with candidates, while (3) with $k = 0$ puts it above every filled cell in the exact score decomposition. Attention thus selects one of the dead-end cells, and the attention OV projection carries its information to the MLP.

Zero candidates shift support toward Revert. For the selected dead-end cell i^*, the OV contribution in (4) reduces to $\mathbf{u}_0 + \mathbf{u}_{\mathrm{pos},i^*}$. In the rollouts of Appendix F.5, the model selects Revert at every observed dead end. The two strongest Revert contributors, neurons 339 and 406, raise their combined mean logit contribution from 0.15 to 16.18 when the selected cell's candidates are cleared in 324 paired states. The original action's top-ten neuron group reduces its mean contribution from 11.92 to 1.07. The model chooses Revert in all 324 cases (Figure 6(b); Appendix F.5).

6 Conclusion

We studied the boundary between a model and its harness in its simplest form: a one-layer, one-head Transformer paired with a simple harness, on Sudoku. Trained by imitating an MRV solver on easy puzzles, the Transformer generalizes from easy to hard puzzles: applied repeatedly, it solves all held-out easy, medium, and hard puzzles and 99.8% of extreme puzzles, and harder puzzles need only more test-time compute. Thanks to its simple structure, we can trace each action: attention selects the cell with the fewest candidates, the MLP favors the teacher's digit, and a dead-end cell shifts the output to Revert. The harness supplies candidate tracking and exact backtracking, and the model supplies the choices. The next question is whether this division of labor extends to complex tasks such as ARC-AGI (Chollet, 2019), whose state, actions, and teacher are harder to specify.

AI USE STATEMENT

We used generative AI tools (large language models) to help revise and edit the text, check for grammatical errors, and check the consistency of the mathematical notation and the validity of derivations. We also used them to help implement, monitor, and debug the experiments. The authors drafted the paper and derived all results themselves, and all reported results are genuine. The authors reviewed all AI-assisted work and take full responsibility for the content of this paper.

ETHICS STATEMENT

This work studies how a small Transformer, paired with a symbolic environment, learns to solve Sudoku puzzles. It involves no human subjects, crowdsourcing, or personal data. All puzzles come from the public Sudoku-Extreme collection (Appendix B.1), which contains only puzzle grids and their solutions. Our analyses aim to clarify how a learned model and its harness share the work of reasoning, and we are not aware of foreseeable ethical concerns arising from this work.

REPRODUCIBILITY STATEMENT

Section 3.1 defines the state, the actions, and the environment transition, and Appendix D gives the environment pseudocode. Section 3.2 specifies the architecture, including the input encoding, the dimensions, and the normalization. Section 3.3 and Appendix B describe the MRV teacher (Algorithm 1), the rollout protocol and success criterion, the puzzle source and difficulty categories, and the generation of training data, including training-set sizes and seeds. Appendix E reports training-error diagnostics, and Section 5 and Appendix F describe the mechanism analyses and intervention protocols. Appendix C collects the notation. We will publicly release the code and all artifacts upon publication, including the data and the trained model.

REFERENCES

Awni Altabaa, Siyu Chen, John Lafferty, and Zhuoran Yang. Unlocking out-of-distribution generalization in transformers via recursive latent space reasoning. *arXiv preprint arXiv:2510.14095*, 2025.

Anthropic. Claude Code. Product page, 2025. URL https://claude.com/product/claude-code.

Artur Back de Luca and Kimon Fountoulakis. Simulation of graph algorithms with looped transformers. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 2319–2363. PMLR, 2024. URL https://proceedings.mlr.press/v235/back-de-luca24a.html.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-Recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=QuqsEIVWIG.

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. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20232–20242, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/7f70331dbe58ad59d83941dfa7d975aa-Abstract-Conference.html.

François Chollet. On the measure of intelligence, 2019. URL https://arxiv.org/abs/1911.01547.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. URL https://transformer-circuits.pub/2021/framework/index.html.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2409.15647.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (SoS): Learning to search in language. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=2cop2jmQVL.

Khashayar Gatmiry, Nikunj Saunshi, Sashank J. Reddi, Stefanie Jegelka, and Sanjiv Kumar. Can looped transformers learn to implement multi-step gradient descent for in-context learning? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 15130–15152. PMLR, 2024. URL https://proceedings.mlr.press/v235/gatmiry24b.html.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=S3GhJooWIC.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, volume 202, pp. 11398–11442. PMLR, 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Michael Hanna, Ollie Liu, and Alexandre Variengien. How does GPT-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. *Advances in Neural Information Processing Systems*, 36:76033–76060, 2023.

Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.

Jianliang He, Leda Wang, Siyu Chen, and Zhuoran Yang. On the mechanism and dynamics of modular addition: Fourier features, lottery ticket, and grokking, 2026. URL https://arxiv.org/abs/2602.16849.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RzYXb5YWBs.

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

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=QF0N3x2XVm.

Tanishq Kumar, Blake Bordelon, Samuel J. Gershman, and Cengiz Pehlevan. Grokking as the transition from lazy to rich training dynamics. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06110.

Asher Labovich. Stability and generalization in looped transformers, 2026. URL https://arxiv.org/abs/2604.15259.

Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul McVay, Michael Rabbat, and Yuandong Tian. Beyond A*: Better planning with transformers via search dynamics bootstrapping. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=SGoVIC0u0f.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.13382.

Laizhen Li, Jiarui Li, Juanjuan Zhao, Kejiang Ye, Ye Li, Cheng zhong Xu, and Xitong Gao. Grow the harness, not the context: From strategy-free scaffolds to reusable specialist agents, 2026. URL https://arxiv.org/abs/2609.26760.

Ziming Liu, Ouail Kitouni, Niklas Nolte, Eric J. Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://arxiv.org/abs/2205.10343.

Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=zDiHoIWa0q1.

Ryan Lopopolo. Harness engineering: leveraging Codex in an agent-first world. OpenAI blog post, February 2026. URL https://openai.com/index/harness-engineering/.

Kaifeng Lyu, Jikai Jin, Zhiyuan Li, Simon S. Du, Jason D. Lee, and Wei Hu. Dichotomy of early and late phase implicit biases can provably induce grokking. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2311.18817.

Leland McInnes, John Healy, and James Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Mohamad Amin Mohamadi, Zhiyuan Li, Lei Wu, and Danica J. Sutherland. Why do you grok? A theoretical analysis on grokking modular addition. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 35934–35967. PMLR, 2024. URL https://proceedings.mlr.press/v235/mohamadi24a.html.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=9XFSbDPmdW.

Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 16–30, 2023b.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic without algorithms: Language models solve math with a bag of heuristics. In *International Conference on Learning Representations*, volume 2025, pp. 55939–55965, 2025.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

OpenAI. Codex. Product page, 2025. URL https://openai.com/codex/.

Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://papers.nips.cc/paper_files/paper/2018/hash/b9f94c77652c9a76fc8a442748cd54bd-Abstract.html.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. URL https://arxiv.org/abs/2201.02177.

Prithvi Rajasekaran. Harness design for long-running application development. Anthropic Engineering blog post, March 2026. URL https://www.anthropic.com/engineering/harness-design-long-running-apps.

Zirui Ren and Ziming Liu. Are your reasoning models reasoning or guessing? A mechanistic analysis of hierarchical reasoning models. *arXiv preprint arXiv:2601.10679*, 2026.

Leo Raphael Rodrigues and Jian Kang. Dissecting hierarchical reasoning models: A mechanistic study. In *Mechanistic Interpretability Workshop at ICML 2026*, 2026. URL https://openreview.net/forum?id=x7a5aCnTzI.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635. PMLR, 2011. URL https://proceedings.mlr.press/v15/ross11a.html.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2020.

Abulhair Saparov, Srushti Pawar, Shreyas Pimpalgaonkar, Nitish Joshi, Richard Yuanzhe Pang, Vishakh Padmakumar, Seyed Mehran Kazemi, Najoung Kim, and He He. Transformers struggle to learn to search. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9cQB1Hwrtw.

Grigory Sapunov. Universal transformers need memory: Depth-state trade-offs in adaptive recursive reasoning. *arXiv preprint arXiv:2604.21999*, 2026.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2502.17416.

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, pp. 56674–56702, 2024. doi: 10.52202/079017-1805. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/67b31ca159553d5593e62d7b998d63ea-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Yuandong Tian. $\mathbf{li_2}$: A framework on dynamics of feature emergence and delayed generalization. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/7ce5da35e01cfa8d303c2dc71e61a470-Abstract-Conference.html.

Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency, 2023. URL https://arxiv.org/abs/2309.02390.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model, 2025a. URL https://arxiv.org/abs/2506.21734.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6545–6554. PMLR, 2019. URL https://proceedings.mlr.press/v97/wang19e.html.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 50208–50232. PMLR, 2024. URL https://proceedings.mlr.press/v235/wang24h.html.

Zhiwei Wang, Yunji Wang, Zhongwang Zhang, Zhangchen Zhou, Hui Jin, Tianyang Hu, Jiacheng Sun, Zhenguo Li, Yaoyu Zhang, and Zhi-Qin John Xu. Understanding the language model to solve the symbolic multi-step reasoning problem from the perspective of buffer mechanism. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 16446–16474, 2025b. URL https://aclanthology.org/2025.findings-emnlp.893/.

Lilian Weng. Harness engineering for self-improvement. Lil'Log (lilianweng.github.io) blog post, July 2026. URL https://lilianweng.github.io/posts/2026-07-04-harness/.

Lyumin Wu, Chenyang Zhang, and Yuan Cao. Looped transformers with layer normalization provably learn the power method, 2026. URL https://arxiv.org/abs/2606.00605.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=EytBpUGB1Z.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024a. URL https://openreview.net/forum?id=mXpq6ut8J3.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024b. URL https://arxiv.org/abs/2311.12424.

Tong Yang, Yu Huang, Yingbin Liang, and Yuejie Chi. Agentic transformers provably learn to search via reinforcement learning, 2026. URL https://arxiv.org/abs/2606.00183.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Alex Zhang and Omar Khattab. Language model harnesses are compositional generalizers. Blog post, July 2026. URL https://alexzhang13.github.io/blog/2026/harness/.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

The supplementary material provides extended related work (Appendix A), the Sudoku task, environment, MRV teacher, rollouts, and training data (Appendix B), a notation reference (Appendix C), environment pseudocode (Appendix D), and training-error diagnostics (Appendix E), followed by the unified mechanistic analysis in Appendix F.

A Detailed related work

A.1 Recursive reasoning and looped Transformers

Recursive reasoning reuses a learned computation over multiple steps, allowing computation to grow without a corresponding increase in parameter count. Looped Transformers implement this idea by repeatedly applying shared blocks to evolving hidden representations, as in Universal Transformers (Dehghani et al., 2019). Theoretical work establishes the algorithmic potential of this recurrence: looped Transformers can execute programs (Giannou et al., 2023) and, with graph-access attention, simulate breadth- and depth-first search (Back de Luca & Fountoulakis, 2024). Studies of learned iterative procedures provide complementary evidence, including parameter-efficient in-context data fitting (Yang et al., 2024b), multi-step preconditioned gradient descent in specialized linear architectures (Gatmiry et al., 2024), and power iteration with layer normalization (Wu et al., 2026).

A central question is whether additional iterations yield useful computation beyond the training regime. Saunshi et al. (2025) connect looping to multi-step reasoning, while Fan et al. (2025) improve generalization to longer inputs by adapting recurrence. Stable extrapolation also depends on architecture and training: Bansal et al. (2022) use input recall and progressive training to prevent degradation with additional iterations, and Altabaa et al. (2025) combine input-adaptive recurrence, algorithmic supervision, discrete latent states, and error correction for compositional generalization. At language-model scale, recurrent depth supplies additional inference-time computation (Geiping et al., 2025; Zhu et al., 2025); token-level routing and budget-conditioned depth further allow computation to be allocated adaptively (Bae et al., 2025; Jeddi et al., 2026).

Sudoku provides a closely related setting for studying compact recursive models. The Hierarchical Reasoning Model (HRM) and Tiny Recursive Model (TRM) use compact recursive networks with deep supervision on Sudoku, mazes, and ARC (Wang et al., 2025a; Jolicoeur-Martineau, 2025). Related Sudoku studies examine memory-token/depth trade-offs (Sapunov, 2026) and the effects of input recall and normalization on loop stability (Labovich, 2026).

Like these works, we ask how a small model can use repeated computation to generalize beyond its training horizon. The closest works are HRM and TRM, which are evaluated on the same Sudoku-Extreme collection. Our work is also closely related to Altabaa et al. (2025), who study out-of-distribution generalization through recurrence. Our work differs from them in three ways. First, in our work, recursive computation occurs through interaction with a deterministic environment rather than inside the network: the same Transformer processes each updated grid and candidate state, while the environment maintains the search history and propagates constraints. Second, our Transformer carries no hidden state between decisions and is trained on single teacher decisions. HRM and TRM, in contrast, iterate a latent state that is trained end to end to produce the complete solution. Third, HRM and TRM iterate multi-layer networks, and Altabaa et al. (2025) analyze a two-layer model, whereas our Transformer has one layer and one head, which lets us identify the attention and MLP computations behind each decision. This setting lets us study how a learned local decision rule composes over long search trajectories and connects our work to the agent harnesses discussed next.

A.2 Reasoning with agent harnesses

Agent harnesses divide reasoning between a learned model and a surrounding system that executes actions, maintains state or memory, and provides feedback. ReAct interleaves reasoning with environment interaction (Yao et al., 2023), while Reflexion stores feedback-derived reflections in external memory to improve subsequent attempts without updating model weights (Shinn et al., 2023). This division makes the surrounding interface part of the reasoning system: SWE-agent improves performance through purpose-built navigation, editing, and execution interfaces (Yang et al., 2024a), and CodeAct uses executable actions and interpreter feedback (Wang et al., 2024). Harnesses can also

organize computation across decisions. External tree search explores alternatives and backtracks in web tasks (Koh et al., 2025), while Growing Harness moves recurring control decisions into reusable executable code (Li et al., 2026).

This perspective raises a mechanistic question: what does the model itself learn when it acts within such a system? Particularly related to our study, Yang et al. (2026) analyze how a two-head Transformer learns exploration and backtracking from sparse terminal rewards on hidden k-ary trees. The heads retrieve tried actions and failure labels from interaction history; under a structured parameterization and depth-wise curriculum, training on depth-1 and depth-2 trees yields generalization to deeper full trees. Their analysis connects successful interaction to specific learned computations.

Our work studies this question with a one-layer, one-head Transformer trained on MRV teacher trajectories, and differs from Yang et al. (2026) in three ways. First, they train with reinforcement learning from sparse terminal rewards, whereas our Transformer is trained by imitating a teacher's actions. Second, their model reads the interaction history in context: its heads retrieve tried actions and failure labels from earlier steps. In our work, the model does not read the interaction history. It selects a cell–digit placement or requests REVERT, while a deterministic environment executes the action, propagates constraints, and maintains the backtracking stack; the model receives only the current grid and candidate state and carries no hidden state between decisions. Third, they obtain generalization from depth-1 and depth-2 trees to deeper trees with a structured parameterization and a depth-wise curriculum. We train a standard one-layer Transformer on short, easy Sudoku trajectories, without a curriculum, and study generalization to much longer searches on harder puzzles; we then identify the attention and MLP computations that support each decision.

A.3 LEARNING TO SOLVE AND TO SEARCH BY IMITATION

Neural networks have been trained to solve Sudoku directly, for example with recurrent relational message passing (Palm et al., 2018) or a differentiable satisfiability layer (Wang et al., 2019). Transformers can also learn search procedures from traces of a classical algorithm: Searchformer imitates the execution traces of A* search (Lehnert et al., 2024), Stream of Search trains a language model on serialized searches that include backtracking (Gandhi et al., 2024), and Shah et al. (2024) train a causal language model on Sudoku solving steps ordered by a solver. Learning to search is not automatic: Saparov et al. (2025) find that Transformers struggle to learn graph search as graphs grow. Imitation of a teacher's actions also faces a classical difficulty: small errors move the policy to states the teacher never visited, and errors can compound over long rollouts (Ross et al., 2011).

The closest work is Shah et al. (2024), and our work differs from it in three ways. First, their model generates a complete solving sequence token by token, so the search state lives in its context; our model makes one decision per step from the current state, while the environment carries out propagation and keeps the search history. Second, Shah et al. (2024) report solving accuracy on Sudoku puzzles, whereas we train only on short traces from easy puzzles and test whether the learned rule, applied for thousands of steps, solves hard and extreme puzzles. Third, they use linear probes to show that the candidates of each cell can be decoded from the model's representations; we trace each decision to specific attention and MLP computations and test them with interventions. The second point also bears on compounding errors: we train by plain imitation, without collecting additional states from the model's own rollouts, yet the model reproduces the teacher's action at nearly every visited state, so errors do not accumulate on the evaluated puzzles (Section 4).

A.4 GROKKING AND THE TRANSITION TO GENERALIZATION

Grokking describes delayed generalization after a model has already fit its training data, first observed on small algorithmic datasets by Power et al. (2022), who also show that smaller datasets can require substantially longer optimization to generalize. Liu et al. (2022) relate this behavior to the formation of structured representations and a critical training-set size in a toy setting, while Omnigrok (Liu et al., 2023) examines the roles of weight norm and regularization and extends the phenomenon beyond algorithmic data. Varma et al. (2023) propose that increasing the training set makes memorizing circuits less efficient relative to generalizing circuits, predicting a critical dataset size and demonstrating ungrokking and partial generalization near this boundary.

Theoretical accounts connect grokking to a transition from nearly fixed features to feature learning (Kumar et al., 2024); under large initialization and small weight decay, Lyu et al. (2024) prove a sharp transition from early kernel predictors to later minimum-norm or maximum-margin solutions in homogeneous networks. For modular addition, Mohamadi et al. (2024) establish sample-complexity separations between kernel predictors and suitably regularized quadratic networks, while Tian (2026) analyze lazy, independent, and interactive feature-learning stages and relate sample size to the stability of generalizing features.

Mechanistically, Nanda et al. (2023a) trace grokking in modular addition through memorization, gradual circuit formation, and removal of memorizing components, showing how continuous internal progress can precede abrupt behavioral improvement. He et al. (2026) further characterize grokking in two-layer modular-addition networks as memorization with perturbed Fourier features, followed by rapid feature cleanup under competing loss minimization and weight decay, and a slower final cleanup dominated by weight decay. Their progress measures track phase alignment, frequency sparsity, and parameter norms, revealing that generalization can improve while parameter norms are still growing.

The closest works are Varma et al. (2023), whose critical dataset size parallels our data-size sweep, and Nanda et al. (2023a), who link an abrupt behavioral transition to a learned circuit. Our study differs in two ways. First, we examine a transition across training-set sizes at a fixed optimization budget, rather than over training time, for a policy whose single-step accuracy must be composed over long rollouts; we therefore measure both held-out action accuracy and solve rate. Second, Nanda et al. (2023a) follow the formation of a circuit over training; we characterize the decision mechanism of the final trained model, and how that mechanism forms as the training set grows is left open.

A.5 Mechanistic interpretability

Mechanistic interpretability seeks an algorithmic account of model behavior in terms of specific attention, MLP, and residual-stream computations. The circuits framework decomposes an attention head into a query–key circuit that decides where to attend and an output–value circuit that decides what to copy (Elhage et al., 2021); our analysis uses this decomposition. Small, well-specified tasks have enabled precise accounts of learned computation: induction-head circuits support continuation copying from repeated contexts (Olsson et al., 2022), while Fourier circuits implement modular addition (Nanda et al., 2023a). Beyond arithmetic, Li et al. (2023) show that OthelloGPT internally tracks board state, and Nanda et al. (2023b) identify a linear, player-relative representation whose causal role can be tested through activation interventions. Related circuit-level studies find that attention and MLP components implement greater-than comparisons in GPT-2 (Hanna et al., 2023), while sparse MLP neurons in larger pretrained models implement input-dependent arithmetic heuristics whose combination explains much of their arithmetic behavior (Nikankin et al., 2025). Moving from single-step operations to compositional reasoning, Wang et al. (2025b) find that Transformers solve controlled symbolic multi-step tasks by storing intermediate variables in distinct buffers and selectively retrieving them through query–key matching. Recent studies of hierarchical recurrent reasoners further identify abrupt correction transitions and multiple fixed points (Ren & Liu, 2026), and use recurrent-state interventions, controlled probing, and sparse-autoencoder analyses to characterize iterative latent computation (Rodrigues & Kang, 2026).

Most directly, Altabaa et al. (2025) analyze a recurrent Transformer for modular arithmetic over computation graphs, finding attention mechanisms that route variable identities and retrieve their values, followed by a final MLP in the analyzed two-layer model that performs addition modulo 23; these component-level hypotheses are validated with causal interventions. Our analysis shares this attention-then-MLP structure and differs in two ways. First, the query comes from a fixed $\langle \text{EOS} \rangle$ token, so, unlike induction and retrieval heads (Olsson et al., 2022; Wu et al., 2025), attention does not match content across tokens; it selects the cell with the highest learned score, which reproduces the MRV rule. Second, in Altabaa et al. (2025) the final MLP computes an arithmetic function of the retrieved values; in our model, the MLP maps the selected cell and its candidates to a cell–digit action that follows the teacher's digit priority, and the same attention routes dead-end cells to REVERT.

B Sudoku task and MRV teacher

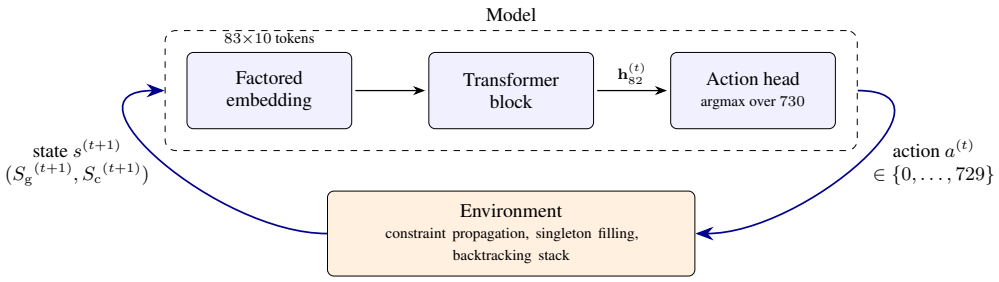


Figure 7: One step of the loop between the Transformer and the environment: the Transformer maps $s^{(t)} = (S_g{}^{(t)}, S_c{}^{(t)})$ to an action $a^{(t)} \in \{0, \ldots, 729\}$, and the environment returns $s^{(t+1)}$.

In this section, we provide details on the Sudoku task and the MRV teacher that generates our training data (Figure 7). In particular, we state the rules of Sudoku and the difficulty categories (Appendix B.1), describe the environment and how its states and actions are encoded for the Transformer (Appendix B.2), explain the MRV heuristic and the teacher (Appendix B.3), define when a rollout succeeds and how we measure it (Appendix B.4), and describe how teacher trajectories become training data (Appendix B.5).

B.1 The Sudoku task

In this subsection, we state the rules of Sudoku, define the terms used throughout the paper, and describe the puzzles and their difficulty categories.

Sudoku rules. A Sudoku grid has 81 cells arranged in 9 rows, 9 columns, and 9 disjoint 3×3 boxes. We call each row, column, and box a *unit*. A solved grid contains every digit $1, \ldots, 9$ exactly once in each of the 27 units. Each cell therefore shares a unit with 20 other cells, its *peers*: 8 in its row, 8 in its column, and 4 more in its box. A puzzle fixes the digits of some cells, the *clues*, and the task is to fill the remaining cells. Every puzzle we use has a unique solution, as guaranteed by the Sudoku-Extreme collection,[1] and is paired with that solution as its reference. Figure 8(a) shows an example puzzle.

Candidates, singletons, and dead ends. While a puzzle is being solved, most empty cells cannot take every digit. The *candidates* of an empty cell are the digits that can still be placed there: digits not already used by any of its peers and not eliminated by an earlier failed attempt. An empty cell with exactly one candidate is a *singleton*, whose digit is forced. An empty cell with no candidates, a *dead-end cell*, cannot be filled at all, so a grid containing one is a *dead end*: no completion of it is consistent with the rules, and some earlier choice must be undone.

Puzzle source and difficulty categories. To test how far imitation on easy puzzles carries over, we grade puzzles into four difficulty categories and evaluate on each.

All puzzles come from the public Sudoku-Extreme collection,[2] which aggregates instances from the benchmark sets of the `tdoku` solver[3] and from the hardest-sudoku thread of the `enjoysudoku` forum,[4] and which pairs every puzzle with its solution.

We assign difficulty using the backtrack count of a separate MRV preprocessing solver with row-major tie-breaking and singleton constraint propagation. Each failed candidate assignment counts as one backtrack. After excluding zero-backtrack puzzles and preprocessing timeouts, we group puzzles into easy (1–9), medium (10–99), hard (100–999), and extreme (at least 1,000 backtracks). These

[1] https://huggingface.co/datasets/sapientinc/sudoku-extreme#dataset-characteristics

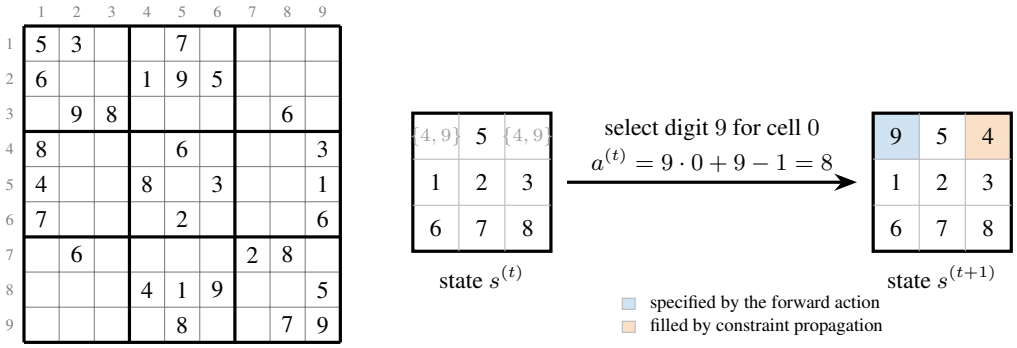
[2] https://huggingface.co/datasets/sapientinc/sudoku-extreme

[3] https://github.com/t-dillon/tdoku

[4] http://forum.enjoysudoku.com/the-hardest-sudokus-new-thread-t6539-600.html

categories describe the preprocessing solver's search cost and need not match the teacher's number of REVERT actions. In the teacher runs of Appendix F.2.5 (1,000 puzzles per category), no puzzle is solved by the environment's constraint propagation alone.

We construct our own training and evaluation partitions within the difficulty-filtered training pools. The generalization evaluations and data-size experiments use puzzles disjoint from their respective training sets. Separately, we use 1,000 puzzles per difficulty to measure the MRV teacher's search lengths and backtracking counts. These descriptive samples may overlap with training puzzles.



(a) Example puzzle (b) One environment transition

Figure 8: **The task and one transition. (a)** An example puzzle. Row and column indices in gray run $1, \ldots, 9$, and the cell in a given row and column has index $i = 9(\text{row} - 1) + (\text{col} - 1)$. **(b)** An independent example showing the upper-left 3×3 box in one environment transition. The selected forward action specifies digit 9 for cell 0. The environment places the digit, after which constraint propagation fills the resulting singleton with digit 4. The shading distinguishes the two updates.

B.2 ENVIRONMENT AND INPUT ENCODING

In this subsection, we describe the environment that carries out the consequences of each action, and the encoding that lets a Transformer act as the policy that chooses the actions.

State representation. The state at step t is the pair $s^{(t)} = ({S_g}^{(t)}, {S_c}^{(t)})$ (Section 3.1). The grid ${S_g}^{(t)} \in \{0, \ldots, 9\}^{81}$ stores each cell's digit, with 0 for an empty cell. The candidate state ${S_c}^{(t)} \in \{0, 1\}^{81 \times 9}$ stores, for each cell, one row of nine bits marking its candidates. For the highlighted cell of Figure 1, ${S_g}^{(t)}[10] = 0$ records that the cell is empty, and ${S_c}^{(t)}[10, :] = (1, 1, 0, 1, 0, 0, 0, 0, 0)$ records its candidates $\{1, 2, 4\}$. The candidate state is part of the state for a specific reason. The policy keeps no memory from one step to the next, and the grid alone does not record which digits were eliminated by earlier failed attempts. Without ${S_c}^{(t)}$, two states with the same grid but different search histories would look identical. With it, the state contains everything needed to choose the next action.

Action space. The policy can do one of two things. A *forward action* (i, d) tries digit d in empty cell i; it is legal when d is a candidate of i. REVERT asks the environment to undo the most recent choice. We number the 729 forward actions as $a = 9i + d - 1 \in \{0, \ldots, 728\}$ and reserve $a = 729$ for REVERT, so every action is a single index. For example, trying digit 1 in cell 10 is action $a = 9 \cdot 10 + 1 - 1 = 90$.

Transitions and constraint propagation. The environment applies each action and prepares the next state (Algorithm 2 in Appendix D). A legal forward action first saves the current state and the choice on a backtracking stack, so that it can be undone. The environment then places the digit and removes it from the candidates of the cell's 20 peers. It then runs *constraint propagation*, Propagate, which fills every singleton with its remaining digit, removes that digit from the singleton's peers, and repeats until no singleton remains. In the constraint-satisfaction literature, removing a placed digit from its peers is *forward checking*; in Sudoku terminology, filling singletons is the *naked-single* rule. The environment applies no other deduction and never guesses: every choice among two or more

candidates is left to the policy. Figure 8(b) shows one forward action followed by one singleton fill. REVERT pops the most recent choice (i, d) from the stack, restores the saved state, removes d from the candidates of cell i, and propagates again. In the example above, if trying 1 in cell 10 later leads to a dead end, reverting that choice restores the empty cell with candidates $\{2, 4\}$. An illegal forward action, or REVERT with an empty stack, leaves the state unchanged. Because propagation runs after every action, the policy never faces a singleton. At every decision point, each empty cell has either no candidates or at least two.

Input encoding for the Transformer. The state is a set of 81 cells, each described by the same ten numbers (its digit and nine candidate bits), so it maps naturally onto a sequence of tokens (Section 3.2). Cell i becomes the token $c_i^{(t)} = (v_i^{(t)}, b_{i,1}^{(t)}, \ldots, b_{i,9}^{(t)})$, holding its digit $v_i^{(t)} = {S_{\mathrm{g}}}^{(t)}[i]$ and its nine candidate bits $b_{i,d}^{(t)} = {S_{\mathrm{c}}}^{(t)}[i, d-1]$. The highlighted cell of Figure 1, for example, becomes the token $c_{10}^{(t)} = (0; 1, 1, 0, 1, 0, 0, 0, 0, 0)$. The model reads the 81 cell tokens between a $\langle$BOS$\rangle$ and an $\langle$EOS$\rangle$ token. Each token is embedded as the sum of a value embedding, nine candidate-bit embeddings, and a positional embedding, scaled by λ, so that the cell's position is available even though all cells share one format. On the output side, the action head produces one logit per action index, and the selected action is the index with the largest logit. Because forward actions are numbered by cell and digit, the logit of index $9i + d - 1$ scores the placement of digit d in cell i. We do not mask illegal actions, so the model must also learn which placements are legal.

B.3 THE MRV TEACHER

In this subsection, we describe the teacher whose actions the model imitates (Section 3.3). In particular, the environment defines which actions are available but not which one to take, and the teacher, a classical solver built on the MRV heuristic, supplies a target action for every state.

Sudoku as backtracking search. Sudoku is a constraint-satisfaction problem with one variable per empty cell, the digits $1, \ldots, 9$ as values, and an all-different constraint on each unit. Backtracking search solves such problems by repeatedly choosing an unassigned variable, trying one of its remaining values, and undoing the most recent choice whenever the partial assignment can no longer be completed. The environment of Appendix B.2 supplies exactly the operations this search needs: forward actions assign values, and REVERT undoes the most recent assignment. Each forward step involves two decisions: *which* variable to assign next and *which* value to try first. Neither decision affects whether the search finds a solution. Because failed values are removed and the search backs up to earlier choices, no value is discarded until the search below it has failed, so the search cannot miss the solution. The two decisions only change how many steps the search takes.

The MRV heuristic. The minimum-remaining-values (MRV) heuristic makes the first decision. It chooses the variable with the fewest remaining values, here the empty cell with the fewest candidates (Haralick & Elliott, 1980; Russell & Norvig, 2020). It is also known as the *fail-first* principle. A cell with few candidates is the one most likely to fail, so choosing it first exposes a wrong earlier choice after few steps and keeps the search tree narrow. A cell with two candidates, for example, splits the search into only two branches. MRV is a heuristic in the usual sense: it guides the search order without changing the result. Its extreme case is a dead-end cell, which has no candidates and fails immediately; this is where the teacher backtracks.

The teacher policy. Our teacher combines MRV with a deterministic lowest-index tie-break, the fixed digit priority $1, 2, \ldots, 9$ for the second decision, and the environment's propagation and backtracking (Algorithm 1). The result is a complete and deterministic solver. Every state has exactly one teacher action, and for every solvable puzzle the teacher's trajectory ends at the solution. Figure 9 shows one MRV decision and the dead end it leads to.

The teacher's search across difficulty categories. The difficulty categories of Appendix B.1 describe the preprocessing solver's search, not the teacher's. Table 1 reports the teacher's search on 1,000 puzzles per category, the sample analyzed in Appendix F.2.5. The teacher's cost grows sharply with difficulty: the median number of forward decisions rises from 7 on easy puzzles to 791

Algorithm 1 MRV teacher action for a state (S_g, S_c).

Here i_{MRV} and d_{MRV} denote the teacher-selected cell and digit; i^* is reserved for the attention argmax in Section 5.1.

1. Let $\mathcal{E} = \{i : S_g[i] = 0\}$ be the empty cells and $|S_{c,i}|$ the candidate count of cell $i \in \mathcal{E}$.
2. If $|S_{c,i}| = 0$ for some $i \in \mathcal{E}$ (a dead end), return REVERT.
3. Let $k_{\min} = \min_{i \in \mathcal{E}} |S_{c,i}|$ and select $i_{\mathrm{MRV}} = \min\{i \in \mathcal{E} : |S_{c,i}| = k_{\min}\}$, the lowest-index cell with the fewest candidates.
4. Select the first candidate of i_{MRV} in the priority order $1, 2, \ldots, 9$: $d_{\mathrm{MRV}} = \min S_{c,i_{\mathrm{MRV}}}$.
5. Return the forward action $a = 9i_{\mathrm{MRV}} + d_{\mathrm{MRV}} - 1$.

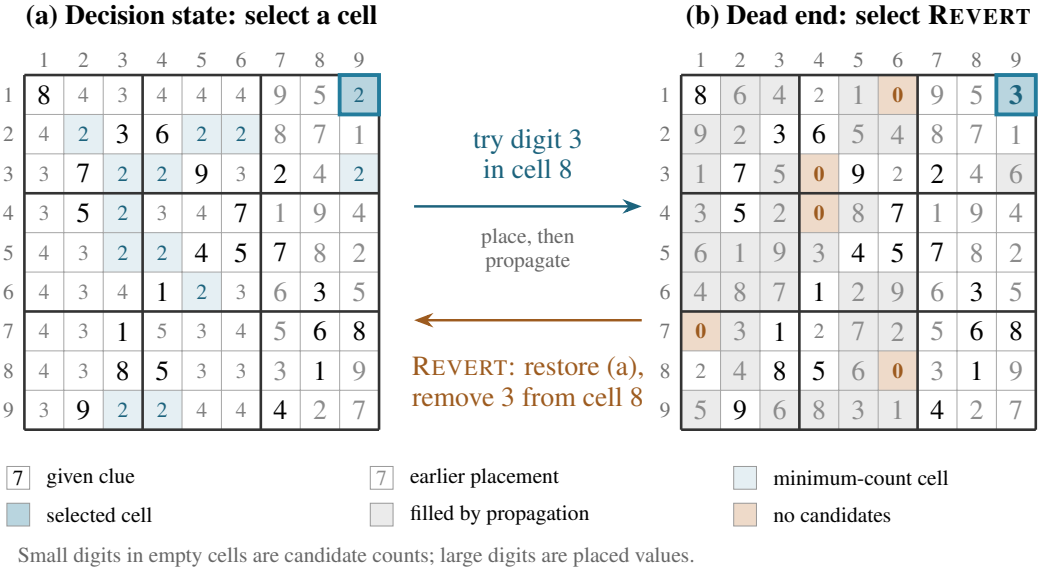


Figure 9: **One MRV decision and the dead end it leads to**, on the puzzle of Figure 1 after six earlier forward decisions by the teacher. **(a)** Small digits give each empty cell's candidate count. Thirteen cells tie at the minimum of two candidates (light blue). The teacher selects the lowest-index one, cell 8 in row 1 and column 9 (dark blue), whose candidates are $\{3, 6\}$, and tries digit 3. **(b)** After placing 3, constraint propagation fills 32 more cells (gray), and five empty cells are left with no candidates (orange). The state is a dead end, so the teacher selects REVERT. The environment restores state (a) and removes 3 from cell 8. The remaining candidate 6 is then a singleton, which propagation fills. On this puzzle, the teacher reaches the solution after 928 forward decisions and 922 REVERT actions.

on extreme puzzles. Within each category the cost varies widely, and the ranges overlap. A few easy puzzles need thousands of teacher decisions, while 11% of extreme puzzles need at most 200.

The table also shows how much of the search is spent on failed branches. In backtracking search, every forward decision that fails is later undone by exactly one REVERT. The difference between the two counts is therefore the number of decisions that survive on the path to the solution. This number is small: 3.1, 4.0, 5.0, and 5.4 decisions per puzzle on average, from easy to extreme. Constraint propagation fills the remaining cells. The same accounting explains why REVERT targets make up 45.1% of the training pairs (Appendix B.5): almost every forward decision in a teacher trajectory is eventually reverted.

What the imitating Transformer must get right. Separating the policy's choices from the environment's bookkeeping clarifies which mistakes matter. The loop keeps the completeness of backtracking search as long as the policy selects a legal forward action at every non-dead-end state and REVERT at every dead end. An illegal action stalls the rollout (Appendix B.4). An unnecessary REVERT removes a digit that may belong to the solution, which can make the solution unreachable. Choosing

Table 1: Length of the MRV teacher's search on 1,000 puzzles per difficulty, without any action cap. Forward decisions include those on failed branches. The last column gives the share of puzzles whose search exceeds the 200 forward decisions used to cap the training trajectories.

	Forward decisions per puzzle			REVERT per puzzle	Share above
Difficulty	Median	Mean	Max	Mean	200 forward
Easy	7	27	5,196	24	2.7%
Medium	26	76	3,430	72	8.3%
Hard	231	312	6,939	307	56.3%
Extreme	791	1,222	144,931	1,216	88.8%

a different legal cell or digit than the teacher, in contrast, only changes how many steps the search takes, and hence the rollout budget needed (Section 4).

B.4 ROLLOUTS AND SUCCESS

In this subsection, we define when a rollout succeeds and the metrics we use to compare the model with the teacher. A rollout is produced by repeating the loop of Appendix B.2 with a policy, either the teacher or the trained model.

Rollout outcomes. A rollout starts from the puzzle after initial constraint propagation and alternates policy actions with environment transitions. It ends in one of three ways:

- **Solved:** the grid is complete and matches the reference solution.
- **Completed incorrectly:** the grid is complete but differs from the reference solution. Every placement made through legal actions and propagation respects the rules, so a complete grid is always a valid Sudoku solution. This outcome can therefore occur only if the puzzle has more than one solution.
- **Budget exhausted:** the number of selected actions reaches the rollout budget $T_{\max}$ before either of the above.

The budget counts every selected action: forward actions, REVERT, and illegal forward actions. Constraint propagation is part of each transition and is not counted. Since an illegal action leaves the state unchanged, a model that selects actions greedily then selects the same action again, so a single illegal action stalls the rollout until the budget is exhausted.

Evaluation metrics. Section 4 uses three metrics. The first two measure agreement with the teacher, and the third measures success:

- **Next action accuracy:** the fraction of evaluated states on which the model selects the teacher's action. The states are fixed in advance, either from the training data or from teacher trajectories on held-out puzzles.
- **Rollout agreement:** the fraction of steps along the model's own rollouts at which its action equals the teacher's action for the state actually reached. After a disagreement, the rollout continues from the model's state, and the teacher's action is recomputed there.
- **Solve rate:** the fraction of puzzles solved within a given budget $T_{\max}$. It is the only metric scored against the reference solution, and it depends on the budget (Figure 3a).

The three metrics are linked by the argument of Appendix B.3. The teacher solves every puzzle given enough budget, so a model that selects the teacher's action at every step solves exactly the puzzles the teacher solves within the same budget.

B.5 TRAINING DATA GENERATION

In this subsection, we describe how teacher trajectories become supervised training pairs, and give the training settings.

Teacher trajectories as training pairs. We run the teacher on $5{,}000$ puzzles from the easy training split, with a cap of 200 forward actions per puzzle. Each forward state is paired with the teacher's action, including actions on paths that later reach a dead end. For each REVERT action, we reconstruct the dead-end state before the environment processes the action and pair it with REVERT. The resulting dataset contains $158{,}215$ pairs, including $71{,}285$ REVERT targets (45.1%). We randomly reserve $15{,}821$ pairs for validation and use the remaining $142{,}394$ for training. This pair-level split is used only to monitor optimization.

Symmetry augmentation. Sudoku's rules do not change under relabeling the digits, rotating the grid, or reflecting it, so each training pair stands for many equivalent pairs. We exploit this as follows. Each time a training pair is drawn, we apply a uniformly sampled digit permutation with probability $1/2$, select uniformly among rotations by $0°$, $90°$, $180°$, or $270°$, and apply a reflection with probability $1/2$, choosing the horizontal or vertical axis uniformly. Every training draw enters this procedure; the digit-permutation and reflection decisions are independent, and the combined transformation can be the identity. We apply the same transformations to the grid and the candidate state. The teacher's action, however, is not invariant: the transformation can change both the lowest-index tie-break and the digit priority. For a forward pair we therefore recompute the MRV target on the transformed state, while a REVERT target remains unchanged. We apply these transformations only to training pairs.

Training objective and optimization. We train the model from scratch using unweighted cross-entropy over the 730 action logits, with hard targets and no label smoothing. We train for $1{,}000$ epochs with a batch size of 256 using AdamW, discarding the final incomplete batch, which gives 556 optimizer steps per epoch and $556{,}000$ steps in total. We use a 100-step linear warmup from 10^{-6} to 10^{-4}, followed by cosine decay to 10^{-6}, with weight decay 10^{-2} and gradient clipping at 1.0. The forward computation uses `bfloat16`.

Data-size experiments. The data-size experiment of Section 4 trains separate models on subsets of the teacher states. Each point of Figure 3(b,c) averages the non-diverged runs among four seeds. Each seed draws its own training subset from the pool of teacher states of one difficulty, and within a seed the subsets are nested across N. Every model trains for $20{,}000$ optimizer steps without augmentation and is evaluated on $14{,}960$ held-out states from 500 puzzles disjoint from all training puzzles, and by rollouts on 100 of these puzzles. Six of the 160 seeded runs diverged during training, reaching at least 94% training accuracy before the loss rose, and are excluded; every other run fits its training set to at least 99.8%.

C NOTATION

In this section, we collect the symbols used in the paper. In particular, we group them by where they are introduced: the task and environment (Section 3.1), the Transformer (Section 3.2), the mechanism analysis (Section 5), and the mechanistic appendix (Appendix F). The last column of each table gives the defining section.

Indexing conventions. Rows and columns are numbered from 1 to 9, and the cell in a given row and column has index $i = 9(\text{row} - 1) + (\text{col} - 1) \in \{0, \dots, 80\}$. Digits are $d \in \{1, \dots, 9\}$, and the candidate state stores digit d in column $d - 1$. The superscript (t) marks the decision step and is omitted in Section 5, which analyzes a single decision. Token position 0 is $\langle \text{BOS} \rangle$, cell i occupies position $i + 1$, and position 82 is $\langle \text{EOS} \rangle$. Per-cell quantities such as $\mathbf{x}_i$, $\mathbf{p}_i$, $\ell_{82,i}$, and δ_i are indexed by the cell index i, while the subscript 82 always refers to the $\langle \text{EOS} \rangle$ token.

Task and environment. These symbols describe the state, the actions, and how the environment updates the state and the backtracking stack.

Symbol	Meaning	Defined in
$s^{(t)} = (S_{\mathrm{g}}{}^{(t)}, S_{\mathrm{c}}{}^{(t)})$	state at decision step t	§3.1
$S_{\mathrm{g}}{}^{(t)} \in \{0,\dots,9\}^{81}$	grid; $S_{\mathrm{g}}{}^{(t)}[i]$ is the digit in cell i, and 0 marks an empty cell	§3.1
$S_{\mathrm{c}}{}^{(t)} \in \{0,1\}^{81\times 9}$	candidates; $S_{\mathrm{c}}{}^{(t)}[i, d-1] = 1$ exactly when d is a candidate of cell i; $S_{\mathrm{c}}{}^{(t)}[i,:]$ is the row of cell i	§3.1
$S_{\mathrm{c},i}$, $\lvert S_{\mathrm{c},i}\rvert$	candidate set of cell i and its size; an empty cell with $\lvert S_{\mathrm{c},i}\rvert = 0$ is a dead-end cell	§3.1
(i,d); $a = 9i + d - 1$	forward action placing digit d in cell i; its index in $\{0,\dots,728\}$	§3.1
$a = 729$; $a^{(t)}$	REVERT: restore the most recent saved state and exclude the tried digit, then propagate constraints; unchanged if the stack is empty. $a^{(t)}$ is the action selected at step t	§3.1
$U^{(t)}$; τ	backtracking stack; step at which the popped decision $(s^{(\tau)}, i, d)$ was saved	§3.1
$(\widetilde{S_{\mathrm{g}}}, \widetilde{S_{\mathrm{c}}})$	intermediate grid and candidate state after the direct action update, before constraint propagation	§3.1
Propagate	operator that repeatedly fills singleton cells and updates peer candidates, returning the next grid and candidate state	§3.1
$T_{\max}$	rollout budget, counting every selected action	App. B.4
N	training-set size in the data-size experiments	§4

Transformer. These symbols describe how a state is encoded as tokens and mapped to action logits by the one-layer Transformer.

Symbol	Meaning	Defined in
$c_i^{(t)} = (v_i^{(t)}, b_{i,1}^{(t)}, \dots, b_{i,9}^{(t)})$	input token of cell i	§3.2
$v_i^{(t)} = S_{\mathrm{g}}{}^{(t)}[i]$	value of cell i (codes 10, 11 for the special tokens)	§3.2
$b_{i,d}^{(t)} = S_{\mathrm{c}}{}^{(t)}[i, d-1]$	candidate bit of digit d in cell i	§3.2
E_{val}; $E_{\mathrm{cand},d}$	value embedding; candidate embedding of digit d	§3.2
$\mathbf{p}_i$; $\mathbf{x}_i^{(t)}$; λ	positional embedding; input embedding of cell i; its scale $\sqrt{d_{\mathrm{model}}/2}$	§3.2
$\mathbf{H}_0^{(t)}, \mathbf{H}_1^{(t)}, \mathbf{H}_2^{(t)}$	hidden states after the embedding, attention, and MLP	§3.2
$\mathbf{Q}^{(t)}, \mathbf{K}^{(t)}, \mathbf{V}^{(t)}$; W_Q, W_K, W_V, W_O	attention queries, keys, values; their weights and the output weight	§3.2
W_1, W_2, W_3	SwiGLU gate, up, and down projections	§3.2
$\mathbf{h}_{82}^{(t)}$	final hidden state of the ⟨EOS⟩ token	§3.2
W_{act}; $\mathbf{z}^{(t)}$; z_a	action head; its 730 logits; the logit of action a	§3.2
d_{model}, d_{head}, d_{ff}; ϵ	widths 256, 256, 512; RMSNorm constant 10^{-5}	§3.2

Mechanism analysis. These symbols decompose the attention score and the attention output at the ⟨EOS⟩ token, and measure how the MLP supports the selected action.

Symbol	Meaning	Defined in
$\mathbf{q}_{82}$; $\mathbf{r}$	fixed ⟨EOS⟩ query; the query mapped to embedding space	§5.1
$\ell_{82,i}$	attention score from the ⟨EOS⟩ token to cell i	§5.1
b; $V(v_i)$	score baseline shared by all cells; value term, with $V(0) = 0$	§5.1
κ_d; C_i	score term of candidate d; its sum over the candidates of cell i	§5.1
δ_i; $\bar{\mathbf{p}}$	positional term; mean positional embedding	§5.1
α	mean score penalty per candidate, so that $C_i \approx -\alpha\lvert S_{\mathrm{c},i}\rvert$	§5.1
i^*	selected cell: the attention argmax $\arg\max_i \ell_{82,i}$	§5.1
i_{MRV}, d_{MRV}	the teacher's cell and digit (Algorithm 1)	App. B.3
$W_{\mathrm{OV}} = W_O W_V$; $\mathbf{o}_{82}$	OV projection; attention output at the ⟨EOS⟩ token	§5.1
$\mathbf{u}_0$, $\mathbf{u}_{\mathrm{pos},i}$, $\mathbf{u}_{\mathrm{val}}(v_i)$; $\mathbf{\Delta}_d$	shared, positional, filled-value, and candidate-d OV terms	§5.1
a^*; m_{action}	teacher action; action margin	§5.2

Mechanistic appendix. These symbols describe individual MLP neurons and are used only in Appendix F.

Symbol	Meaning	Defined in
j; a_j	MLP neuron index; its SwiGLU activation (not an action)	App. F.4.5
$G_{i,d}$	ten neurons with the largest mean squared activation, conditioned on selected cell i and preferred digit d	App. F.4.5
$\gamma^g_{j,d}$, $\gamma^u_{j,d}$	gate and up coefficients of candidate d for neuron j	App. F.4.6
$\beta^g_{i,j}$, $\beta^u_{i,j}$; $\rho(i, C)$	gate and up baselines at cell i; RMS factor for candidate set C	App. F.4.6

D PSEUDOCODE FOR THE ENVIRONMENT STEP

In the pseudocode, U denotes the current backtracking stack, written $U^{(t)}$ in the main text.

Algorithm 2 Environment transition for one selected action.

Inputs: state $(S_{\mathrm{g}}, S_{\mathrm{c}})$, action $a \in \{0, \ldots, 729\}$, and backtracking stack U. Each selected action counts as one step. The action may come from the teacher or the model; the environment only executes the transition. Each stack entry is a saved state and the choice made from it. The popped entry $((S_{\mathrm{g_{prev}}}, S_{\mathrm{cprev}}), i_{\mathrm{prev}}, d_{\mathrm{prev}})$ is the entry $(s^{(\tau)}, i, d)$ of Section 3.1, while (i, d) here denotes the forward action decoded from the input.

1. If $a = 729$ (REVERT):
2. If U is empty, return $(S_{\mathrm{g}}, S_{\mathrm{c}}, U)$ unchanged.
3. Otherwise pop the top of U to get $((S_{\mathrm{g_{prev}}}, S_{\mathrm{cprev}}), i_{\mathrm{prev}}, d_{\mathrm{prev}})$.
4. Set $S_{\mathrm{g}} \leftarrow S_{\mathrm{g_{prev}}}$, $S_{\mathrm{c}} \leftarrow S_{\mathrm{cprev}}$, and $S_{\mathrm{c}}[i_{\mathrm{prev}}, d_{\mathrm{prev}}{-}1] \leftarrow 0$ (remove the failed digit).
5. Run $\mathrm{Propagate}(S_{\mathrm{g}}, S_{\mathrm{c}})$, then return $(S_{\mathrm{g}}, S_{\mathrm{c}}, U)$.
6. Otherwise (forward action), decode $a \to (i, d)$ where $i = \lfloor a/9 \rfloor$ and $d = (a \bmod 9) + 1$.
7. If $S_{\mathrm{g}}[i] \neq 0$ or $S_{\mathrm{c}}[i, d{-}1] = 0$ (illegal forward action), return $(S_{\mathrm{g}}, S_{\mathrm{c}}, U)$ unchanged.
8. Push $((S_{\mathrm{g}}, S_{\mathrm{c}}), i, d)$ onto U.
9. Set $S_{\mathrm{g}}[i] \leftarrow d$ and $S_{\mathrm{c}}[i, :] \leftarrow 0$.
10. For every cell j in the same row, column, or box as i, set $S_{\mathrm{c}}[j, d{-}1] \leftarrow 0$.
11. Run $\mathrm{Propagate}(S_{\mathrm{g}}, S_{\mathrm{c}})$, then return $(S_{\mathrm{g}}, S_{\mathrm{c}}, U)$.

Propagate iteratively fills any empty cell whose remaining candidate set is a singleton, propagates the placed digit to that cell's row, column, and box, and repeats until no further singletons exist.

E TRAINING DYNAMICS: DISJOINT ERROR COMPONENTS

We distinguish four types of disagreement with the teacher to interpret the training curves in Section 4. At each visited state, the teacher action is recomputed on that same state. A forward action is legal if it selects an empty cell and one of its remaining candidate digits. The four components are mutually exclusive:

Error component	Teacher action	Model action
1. Missed backtracking	REVERT	Any forward action
2a. Unnecessary backtracking	Forward	REVERT
2b. Illegal placement	Forward	Illegal forward action
3. Legal teacher disagreement	Forward	Different legal forward action

An illegal forward action at a teacher-REVERT state belongs only to component 1. Component 3 is not necessarily an error: a legal action that differs from the teacher's can still lead to the solution. Each component is divided by *all evaluated decisions*, so their sum equals one minus rollout agreement with the teacher.



Figure 10: Disjoint error components over training on 500 held-out easy puzzles, using greedy rollouts capped at 200 actions per puzzle. Markers show the nine measured checkpoints; connecting lines are visual guides. **(a)** All four components share the same denominator: all visited decisions, including repeated actions after a stall. **(b)** Component 3 from step 3,260 onward on an expanded vertical scale, with the same denominator. At step 50, every prediction is REVERT and no dead end is visited, so a zero missed-backtracking share does not establish correct dead-end recognition.

Protocol and reconstruction. We use the nine checkpoints of the main run for which both rollout-agreement counts and per-action legality checks were recorded. Each checkpoint is evaluated on its own model-generated trajectories; repeated actions after a stall count as separate decisions. We check that the two agree on puzzle counts, total decisions, matches, illegal actions, and matching forward actions. The number of illegal placements is the number of teacher-forward decisions minus unnecessary backtracks and legal forward actions. Subtracting matching forward actions from the latter yields legal teacher disagreements. The resulting nonnegative integer counts exactly partition all disagreements at every plotted checkpoint.

Observed progression. At step 3,260, both backtracking error counts are zero, but 19,494 of 28,818 decisions (67.65%) are illegal placements and 116 (0.40%) are legal teacher disagreements. Both backtracking components remain zero at the subsequent plotted checkpoints, while illegal placements dominate the remaining disagreement. Legal teacher disagreement falls to zero by step 348,510; all four components are zero at step 440,200. The progression is not monotone: illegal placements increase again at step 108,410.

Takeaway. Backtracking decisions stabilize before forward actions become consistently legal. From step 3,260 onward, illegal placements account for at least 98.2% of the remaining teacher disagreements at every plotted checkpoint with nonzero error. Legal teacher disagreements are already rare, accounting for at most 0.52% of all decisions over these checkpoints. Thus, the main benefit of later training in this run is eliminating illegal placements. Because an illegal placement leaves the state unchanged and is repeated under greedy action selection, eliminating these errors prevents the rollout from stalling and allows the trained model to proceed reliably.

F MECHANISTIC ANALYSIS: DERIVATIONS, EXPERIMENTS, AND LIMITS

This appendix develops the evidence behind Section 5. After defining the shared measurement conventions (Appendix F.1), we follow the computation from attention-based cell selection (Appendix F.2) through OV routing (Appendix F.3) to MLP forward-action selection (Appendix F.4) and the REVERT decision that initiates backtracking (Appendix F.5). Each part presents its derivation, experimental protocol, results, and limitations.

F.1 Shared setup and measurement conventions

All measurements use the trained model evaluated in Section 4. We trace one model decision and omit the time superscript. The action head reads the hidden state at the ⟨EOS⟩ token and produces 730 action logits. We record this token's attention scores, OV outputs, and MLP activations using forward hooks. Because the query is fixed and no normalization precedes attention in this single-block model, the additive input embeddings admit exact QK and OV decompositions.

Numerical conventions and sampling. Unless stated otherwise, model forward passes use bfloat16 on CPU; parameter-level calculations and numerical reconstructions use FP32 or FP64. Comparisons with the bfloat16 computation report their discrepancies. Sample sizes refer to the unit named in each analysis. Decisions from one puzzle and interventions derived from one source state are not independent trials. The analyzed states were not screened for overlap with supervised training data. Held-out source states in the neuron analyses test transfer within the specified cohorts, without establishing a puzzle-disjoint evaluation.

Digit labels and intervention types. The teacher's digit priority is $1 \prec 2 \prec \cdots \prec 9$ (Appendix B.3), so smaller digits are preferred. Digit permutations relabel the entire grid and every candidate mask, preserving Sudoku constraints. Direct candidate-mask edits keep the grid fixed and can produce states that are not reachable under constraint propagation. Internal interventions modify the specified activations or projections and recompute the downstream action logits. Each experiment identifies which procedure it uses.

F.2 Attention: cell selection and validation

We test the cell-selection mechanism of Section 5.1 using exhaustive parameter calculations, sampled model rollouts, candidate-mask interventions, and complete teacher trajectories. The parameter calculations establish score bounds; the state-based measurements show their behavioral scope and exceptions.

F.2.1 Exact attention-logit decomposition

The exact attention-logit decomposition and all four components are defined in (1) and (2). Here we test the stored-weight decomposition and its relation to the finite-precision forward pass. The shared baseline is $b \approx 1{,}079.598$ logits. By definition, $V(0) = 0$, and $C_i = 0$ when no candidate is present. For filled cells, $V(v_i)$ ranges from -680.858 to -663.595 logits; Figure 5c plots the nine magnitudes as empty-cell value advantages. For empty cells, the shared-penalty approximation is

$$\ell_{82,i} \approx b - \alpha|S_{\mathrm{c},i}| + \delta_i, \qquad v_i = 0. \tag{6}$$

The following enumeration quantifies the approximation error and tests whether position can override candidate-count ordering.

F.2.2 Candidate-mask enumeration and the positional bound

For Figure 5a and the residual view in Figure 11, we enumerate all $2^9 = 512$ possible candidate sets using the learned candidate-embedding differences and the query and key projections. The coefficient κ_d in (2) is the logit change from switching candidate bit d from absent to present. For a candidate set $S \subseteq \{1, \ldots, 9\}$, let $C(S) = \sum_{d \in S} \kappa_d$ denote its contribution to the attention logit. For cell i, $C(S) = C_i$ when $S = \{d : b_{i,d} = 1\}$. Each dot in Figure 5a shows $C(S)$ at the exact candidate count $k = |S|$. The line shows the prediction $-\alpha k$, where $\alpha = -\frac{1}{9}\sum_{d=1}^{9} \kappa_d$. The two-candidate group, for instance, contains all $\binom{9}{2} = 36$ pairs: the dot for $\{1, 2\}$ shows $\kappa_1 + \kappa_2$ and the dot for $\{1, 3\}$ shows $\kappa_1 + \kappa_3$, both at $k = 2$. To quantify agreement with the line, we compute the mean within each count group and subtract it from each set's contribution:

$$\overline{C}_k = \frac{1}{\binom{9}{k}} \sum_{\substack{S \subseteq \{1,\ldots,9\} \\ |S|=k}} C(S), \qquad r(S) = C(S) - \overline{C}_{|S|}.$$

Subtracting the group mean thus removes the effect of candidate count and leaves the effect of which digits are candidates. The group mean is exactly $\overline{C}_k = -\alpha k$, and the residuals range from -0.682

to $+0.682$ logits, so the points within each count group overlap at the scale of Figure 5a. Figure 11 displays these residuals explicitly. The mean candidate penalty is $\alpha \approx 196.300$, with standard deviation 0.217 across digits. Across two-candidate sets, the largest value of $-C(S)$ is 393.243 logits. These masks exhaust the binary candidate-mask space for one cell, so the plotted spread describes variation among masks with the same candidate count rather than statistical uncertainty.

For the ordering bound we also compute the full range of $C(S)$ within each count group. For each pair of adjacent counts we subtract the largest candidate contribution at the higher count from the smallest contribution at the lower count. The minimum of these nine gaps is 194.952. Any pair of positions changes the logit difference by at most the full positional range, 193.950, leaving a positive margin of approximately 1.002 logits. This establishes unequal-count ordering for the exact decomposition. It neither guarantees the ordering once the bfloat16 forward computation is applied nor proves lowest-index tie-breaking for every equal-count pair. Figure 5b plots all 81 positional offsets directly from the same learned parameters. Checking the additive formula against complete empty-cell embeddings for all $512 \times 81 = 41{,}472$ mask-position combinations reproduces the logits with maximum absolute error 4.55×10^{-13}.

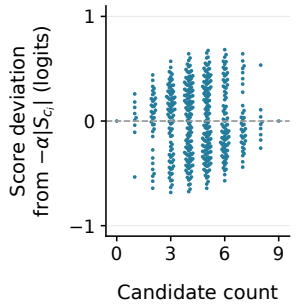


Figure 11: Candidate-identity effects after removing the mean contribution at each candidate count. All 512 masks lie within 0.682 logits of the count-only prediction in (6).

F.2.3 Deriving the empty-versus-filled score bound

We derive the lower bound in Section 5.1 directly from the trained model's embeddings and QK weights, using FP64 for all projections in this subsection. For an empty cell j with two candidates and a filled cell i, (3) gives the exact score difference

$$\ell_{82,j} - \ell_{82,i} = -V(v_i) + C_j + (\delta_j - \delta_i).$$

We bound these three terms separately.

1. **Minimum value advantage: enumerate the nine filled digits.** For each $v = 1, \ldots, 9$, project its embedding difference from the empty value through the fixed $\langle \text{EOS} \rangle$ query and key weights:

$$-V(v) = \frac{\lambda \mathbf{q}_{82}^{\top} W_K \big(E_{\text{val}}(0) - E_{\text{val}}(v)\big)}{\sqrt{d_{\text{head}}}}.$$

   The smallest of these nine values is $A_{\min} = \min_v[-V(v)] = 663.594666$ logits, attained at digit $v = 5$. Hence $-V(v_i) \geq A_{\min}$ for every filled cell. Figure 5c shows the nine advantages. Its archived FP32 projections differ from this FP64 calculation by less than 3×10^{-5} logits.

2. **Maximum candidate penalty: enumerate all 36 candidate pairs.** Using the coefficients κ_d defined in Appendix F.2.2, compute

$$P_2 = \max_{1 \leq d < e \leq 9} \big[-(\kappa_d + \kappa_e)\big] = 393.243429 \text{ logits}.$$

   The largest penalty is the sum of the two largest individual penalties, $196.835256 + 196.408173$. Thus, every two-candidate cell satisfies $C_j \geq -P_2$. This uses the exact candidate coefficients; $2\alpha \approx 392.601$ is the average pair penalty, not its maximum.

3. **Largest positional disadvantage: enumerate the 81 positions.** Project each position using the formula for δ_i in (2). The maximum occurs at cell 0 and the minimum at cell 80, giving

$$R_{\text{pos}} = \max_i \delta_i - \min_i \delta_i = 111.397068 - (-82.553357) = 193.950425 \text{ logits.}$$

Consequently, $\delta_j - \delta_i \geq -R_{\text{pos}}$ for any two cell positions. Subtracting the mean position embedding does not change this range (Figure 5b).

Combining the three inequalities gives

$$\begin{aligned} \ell_{82,j} - \ell_{82,i} &\geq A_{\min} - P_2 - R_{\text{pos}} \\ &\approx 663.594666 - 393.243429 - 193.950425 = 76.400812 > 76. \end{aligned} \tag{7}$$

The displayed values are rounded; the calculation uses the unrounded projections. Rounding the positive term down and both subtracted terms up to three decimal places yields $663.594 - 393.244 - 193.951 = 76.399 > 76$; the main text rounds conservatively to one decimal place.

As an independent check, we project complete cell embeddings for all $81 \times 36 = 2{,}916$ two-candidate empty-cell inputs and all $81 \times 9 = 729$ filled-cell inputs. The smallest empty-cell score minus the largest filled-cell score is 76.400812 logits, agreeing with the bound to within 10^{-9} logits. This enumeration establishes separation in the additive stored-weight score model across all these cell features; it does not require the extreme feature combinations to co-occur on a reachable Sudoku board. The three-candidate cases and finite-precision forward computations are examined in Appendix F.2.5.

F.2.4 Rollout agreement and candidate-mask interventions

Rollout sampling and agreement. We capture the attention weights from the ⟨EOS⟩ token to the 81 cells during greedy rollouts on 64 easy and 64 extreme puzzles, and compare the attention argmax with the teacher-selected cell, including the lowest-index tie-break. The recorded sample covers valid forward decisions for which the teacher also chooses a forward action, not every visited state.

The attention argmax matches the teacher-selected cell on all 565 recorded easy decisions and all 2,154 extreme decisions, including the 560 and 2,107 decisions with multiple minimum-count cells (Figure 12(d)). Mean attention mass on the teacher-selected cell is 0.987 and 0.984.

Candidate-mask interventions. In 200 extreme states in which the teacher selects a forward action, we hold the grid and all other candidate masks fixed and reduce a competing empty cell c_1 to nested candidate subsets of size k, using a fixed ordering of its candidates. If the original minimum at c_0 is m_0, we predict retention of c_0 for $k > m_0$, selection of c_1 for $k < m_0$, and the teacher's positional tie-break for $k = m_0$. The sweep contains 643 conditions, including the unedited endpoint as the $k > m_0$ control. These are candidate-mask interventions, so the edited masks need not be states reachable under constraint propagation.

Under candidate-mask interventions, the attention argmax agrees with the teacher-selected cell after each edit in all 643 conditions: $243/243$ below the original minimum, $200/200$ above it, and $200/200$ at a tie. The edited competitor receives mean mass 1.000 when $k < m_0$ and 2.38×10^{-5} when $k > m_0$ (Figure 12(c)). At $k = m_0$ its mean mass is 0.825 versus 0.174 on c_0. These averages pool different positional configurations and do not imply a comparable split within each state.

Reconstruction and precision checks. We extract the nine individual candidate penalties and the positional contributions directly from the learned parameters. On 300 extreme forward states (16,011 empty-cell observations) we reconstruct the attention row from the full QK computation, measure the accuracy of the shared-penalty approximation in (6), and check count ordering, positional tie-breaking, and suppression of filled cells.

The candidate term is nearly linear in count ($R^2 = 0.999997$). The winner has minimum count in all 300 validation states and the lowest index in all 257 tied states. The largest empty-cell logit exceeds the largest filled-cell logit by at least 25.92 logits in this sample. Attention probabilities reconstructed from the full QK decomposition match the model's attention row to within 3.65×10^{-5}. Rounding in bfloat16 changes individual attention probabilities by as much as 0.113 but leaves the selected cell unchanged on all 150 states in this comparison.

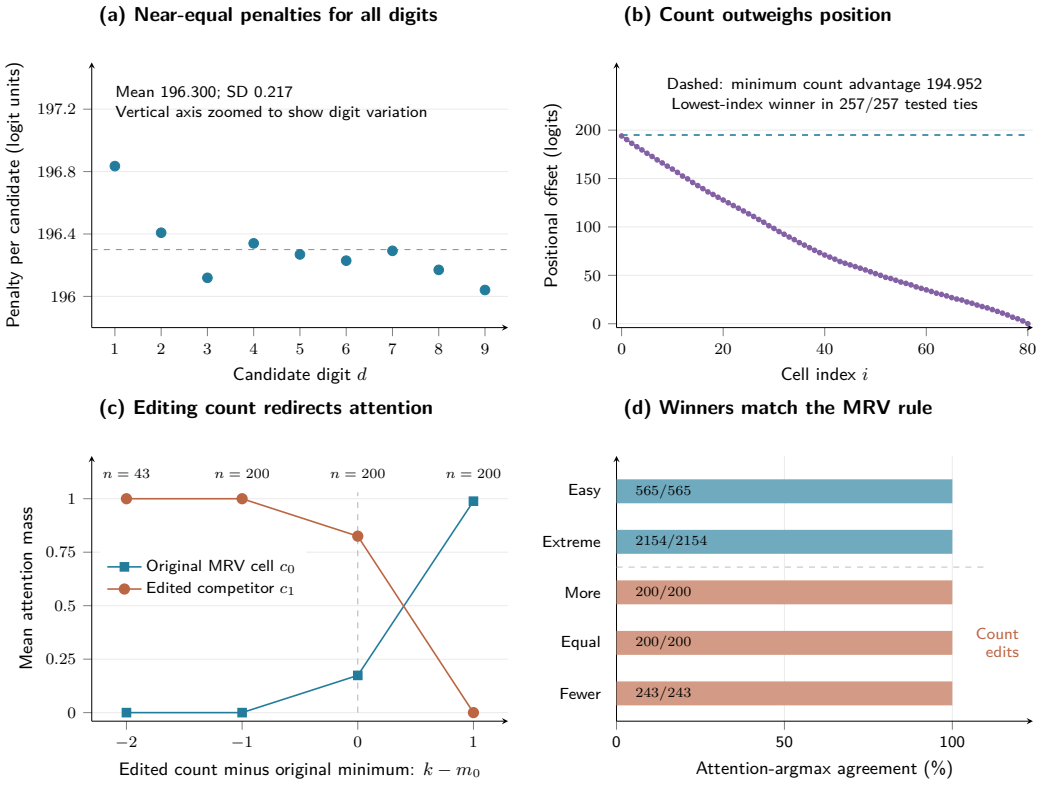


Figure 12: **Evidence for candidate-count selection by attention.** **(a)** Parameter-derived penalties for the nine candidate contributions. The dashed line is their mean and the vertical axis is zoomed. **(b)** All 81 positional offsets $\delta_i - \min_j \delta_j$, compared with the minimum adjacent-count advantage (dashed). These are deterministic parameter-level quantities. **(c)** Candidate-count interventions, averaged over 200 base states, where n gives the number of conditions at each count difference. Lines connect observed discrete conditions, not a fitted continuous response. The tie point averages states with different positional preferences. **(d)** Agreement with the teacher-selected cell in recorded rollouts (blue) and after each edit (orange). The labels *more/equal/fewer* refer to k relative to m_0. Decisions from the same puzzle and edits derived from the same base state are not independent trials.

F.2.5 COMPLETE TEACHER TRAJECTORIES

Candidate counts. We run the MRV teacher on a separate sample of 1,000 puzzles per difficulty, drawn randomly from the same pools as, but independent of, the 500-puzzle sets of Section 4. The teacher selects the lowest-index minimum-candidate cell and uses the fixed ordering for digit selection. The search has no forward-action limit and includes all failed branches and backtracking. All 4,000 final grids match the reference solutions. Table 2 counts every forward decision along these trajectories, including repeated visits after backtracking. Singleton cells are filled automatically during constraint propagation, so they do not appear as one-candidate forward decisions. The 1,619,014 REVERT decisions occur in states containing an empty cell with no candidates and are counted separately. Solved terminal states have no teacher-selected cell. We verify the counts independently from the recorded candidate masks. These are MRV teacher trajectories rather than model rollouts, and their candidate counts alone do not establish filled-cell exclusion by attention in every state.

Table 2: Candidate counts at MRV forward decisions over complete trajectories of 1,000 puzzles per difficulty. Entries give the number of decisions and the percentage within each row. Automatic single-candidate fills and REVERT decisions are excluded.

Difficulty	Two candidates	Three candidates	Four or more
Easy	27,128 (99.776%)	61 (0.224%)	0 (0%)
Medium	76,095 (99.843%)	120 (0.157%)	0 (0%)
Hard	311,140 (99.862%)	430 (0.138%)	0 (0%)
Extreme	1,220,532 (99.914%)	1,056 (0.086%)	0 (0%)
Total	1,634,895 (99.898%)	1,667 (0.102%)	0 (0%)

Rare filled-cell counterexamples. Using the recorded states from these trajectories, we compute the attention logits from the ⟨EOS⟩ query for all 81 cells at every forward decision. The calculation uses the full value, candidate, and positional embeddings, including the input scale λ, rather than the shared candidate-penalty approximation. We find three states in which a filled cell has the highest logit (Table 3). Each is also verified by a full model forward pass. The three states belong to two puzzles and all have minimum candidate count three. They constitute 0.000183% of all forward decisions and $3/1{,}667 = 0.180\%$ of the three-candidate decisions. These are frequencies over the recorded trajectories, not independent-trial estimates or model solving-failure rates.

Table 3: Filled-cell selection exceptions over all recorded MRV forward decisions. An exception occurs when a filled cell receives the highest attention logit. Each difficulty contains 1,000 puzzles.

Difficulty	Forward decisions	Exceptions	Frequency (%)
Easy	27,189	0	0
Medium	76,215	0	0
Hard	311,570	2	0.000642
Extreme	1,221,588	1	0.0000819
Total	1,636,562	3	0.000183

With puzzle and step indices starting at zero, the teacher-selected cell in hard puzzle 532 of this sample at decision step 12 is cell 29 with candidates $\{2, 3, 5\}$, whereas cell 2 is filled with digit 4. Their logit difference decomposes as

$$\ell_{82,29} - \ell_{82,2} = \underbrace{664.369}_{\text{value advantage}} - \underbrace{588.796}_{\text{candidate penalty}} - \underbrace{85.115}_{\text{positional disadvantage}} \approx -9.543,$$

so the filled cell outranks the teacher-selected cell. This illustrates why the absence of four-candidate MRV states alone does not guarantee filled-cell exclusion.

Attention exceptions and the final action. The three filled-cell selections persist under bfloat16, FP32, and FP64 computation. The model selects an illegal forward action in all three states. Rounding in bfloat16 produces two additional equal-count tie-break reversals, both in extreme puzzle 217 at

steps 592 and 1,351. In these states, cell 69 exceeds the teacher-selected cell 68 by 0.289 logits, although both have two candidates. The count-based ordering still holds and only the index tie-break changes. The model nevertheless selects the teacher action in both states. The attention argmax therefore differs from the teacher-selected cell in five of the $1{,}636{,}562$ recorded forward decisions and is not a universally exact description of the final action. Neither of the two special tokens has an attention logit as large as the largest board-cell logit in these decisions. We check the model's final action by a full forward pass on these five states only.

F.2.6 MRV MARGIN DISTRIBUTIONS

We measure the MRV cell's lead over its strongest competitor using both score and attention-probability margins. Let $\ell_{82,\mathrm{BOS}}$ and $\ell_{82,\mathrm{EOS}}$ denote the attention scores from the $\langle\mathrm{EOS}\rangle$ token to the two special tokens. The corresponding attention weights and margins are

$$a_i = \frac{\exp(\ell_{82,i})}{\exp(\ell_{82,\mathrm{BOS}}) + \sum_{j=0}^{80} \exp(\ell_{82,j}) + \exp(\ell_{82,\mathrm{EOS}})},$$
$$m_\ell = \ell_{82,i_{\mathrm{MRV}}} - \max_{j \neq i_{\mathrm{MRV}}} \ell_{82,j}, \qquad m_a = a_{i_{\mathrm{MRV}}} - \max_{j \neq i_{\mathrm{MRV}}} a_j.$$

Here i_{MRV} is the teacher's minimum-candidate empty cell, with ties resolved by the lowest cell index, and a_i is the softmax attention weight assigned to cell i by the $\langle\mathrm{EOS}\rangle$ token. The denominator includes all 81 board cells and the two special tokens; the maxima in the margin definitions compare only board cells. Positive margins indicate MRV selection; negative margins indicate that a competing cell wins.

Figure 5d uses all $1{,}636{,}562$ recorded forward decisions from the complete MRV teacher trajectories in Table 2, including failed branches, unique minima, and count ties. We exclude REVERT and solved terminal states. The decisions are pooled without deduplication across trajectories, so longer trajectories contribute more observations.

Using FP32 attention logits, we compute the teacher-selected cell's logit minus the maximum attention logit among all other 80 cells, including filled cells. The reference cell uses the teacher's lowest-index tie-break even when it loses, so the three negative margins are retained rather than replaced by positive top-two gaps. Attention logits are computed from the trained model's full embeddings. We enumerate the cell logits by candidate mask, value, and position using the fixed $\langle\mathrm{EOS}\rangle$ query, and compare the resulting values with full model forward passes on 26 states covering all detected exceptions and the smallest margins. The cell logits agree exactly in this check.

The median logit margin is 6.70 logits. Three margins are negative, with a minimum of -9.654, corresponding to the filled-cell counterexamples. The probability margin is at least 0.99 in 65.6% of decisions. The logit histogram retains variation that softmax compresses near one in the probability histogram. Probability margins are computed from these logits using a stable softmax over all 83 tokens, including the two special tokens. Values at one include numerical saturation and do not measure the probability that an action is correct. All histogram bins are normalized by the full number of forward decisions, including the negative-margin cases.

F.2.7 SCOPE OF THE ANALYSIS

The candidate-mask interventions establish a response to candidate edits on the evaluated inputs, but they do not establish that attention is necessary for the final action. The exhaustive bound in Appendix F.2.3 applies to two-candidate empty cells in the stored-weight score model. Comparisons of the largest empty- and filled-cell logits on sampled states do not extend that guarantee to every empty cell or to bfloat16 arithmetic.

F.3 OV ROUTING: DECOMPOSITION AND REPRESENTATION GEOMETRY

The OV path carries the selected cell's location and candidate information to the MLP. We give the vector definitions underlying (4), verify the decomposition, and compare the geometry with the distinctions required by the teacher.

F.3.1 VECTOR CONTRIBUTIONS AND GEOMETRY

The additive input structure that separates the attention scores also separates the information written by each cell. Let $W_{\mathrm{OV}} = W_O W_V$ denote the head's value projection followed by its output projection. Using the empty-value embedding and the nine absent-candidate embeddings as a reference, define the vector contributions

$$\mathbf{u}_0 = \lambda W_{\mathrm{OV}}\Big(E_{\text{val}}(0) + \sum_{d=1}^{9} E_{\text{cand},d}(0)\Big), \qquad \mathbf{u}_{\text{pos},i} = \lambda W_{\mathrm{OV}}\mathbf{p}_i,$$

$$\mathbf{u}_{\text{val}}(v) = \lambda W_{\mathrm{OV}}\big(E_{\text{val}}(v) - E_{\text{val}}(0)\big), \qquad \mathbf{u}_{\text{cand},i} = \lambda W_{\mathrm{OV}} \sum_{d=1}^{9}\big(E_{\text{cand},d}(b_{i,d}) - E_{\text{cand},d}(0)\big).$$

Here $\mathbf{u}_0$ is shared by all cells, and $\mathbf{u}_{\text{pos},i}$ is the contribution from cell i's position. These definitions give the exact per-cell contribution in (4). For an empty selected cell i^*, $\mathbf{u}_{\text{val}}(0) = 0$, so its write is $\mathbf{u}_0 + \mathbf{u}_{\text{pos},i^*} + \mathbf{u}_{\text{cand},i^*}$: a common baseline plus cell position and candidate content. For a filled cell, $\mathbf{u}_{\text{cand},i} = 0$ and the value contribution remains.

The $\langle\text{EOS}\rangle$ token receives an attention-weighted sum of the OV contributions from all 83 tokens. Let $\mathbf{o}_{82}$ denote this output before residual addition and normalization. To connect the decomposition to its MLP input, we use the attention concentration measured in Appendix F.2.6. As shown by the attention-probability margins in Figure 5d, its attention is typically concentrated on its argmax i^*. In the recorded teacher trajectories, this cell coincides with the teacher's cell i_{MRV} in all but five of 1,636,562 forward decisions under bfloat16 computation (Appendix F.2.5). The softmax-weighted output can then be approximated by that cell's contribution:

$$\mathbf{o}_{82} \approx W_{\mathrm{OV}}\mathbf{x}_{i^*} = \mathbf{u}_0 + \mathbf{u}_{\text{pos},i^*} + \mathbf{u}_{\text{cand},i^*} \quad \text{when } a_{i^*} \approx 1 \text{ and } v_{i^*} = 0.$$

The MLP must distinguish both cell locations and candidate digits from this combined output. We examine the geometry of their OV vectors to see how these features are represented. For each digit d, define its candidate contribution as $\mathbf{\Delta}_d = \lambda W_{\mathrm{OV}}(E_{\text{cand},d}(1) - E_{\text{cand},d}(0))$, so that $\mathbf{u}_{\text{cand},i} = \sum_{d \in S_{\mathrm{c},i}} \mathbf{\Delta}_d$. We compare the directions of these components using cosine similarity. Figure 13(a) shows pairwise cosine similarities among the shared baseline $\mathbf{u}_0$, the nine filled-value vectors $\mathbf{u}_{\text{val}}(v)$ ($v = 1, \ldots, 9$), the nine candidate vectors $\mathbf{\Delta}_d$ ($d = 1, \ldots, 9$), and the 81 position vectors $\mathbf{u}_{\text{pos},i}$ ($i = 0, \ldots, 80$). We compute these cosines directly from the learned weights. The same baseline-relative decomposition reconstructs all 512 candidate sets and nine filled digits at all 81 positions with maximum vector-coordinate error 3.91×10^{-14}. The common input scale cancels in cosine similarity.

Position vectors have low mean absolute cosine with one another (0.137), the shared baseline (0.124), filled-value vectors (0.124), and candidate vectors (0.089). Candidates 8 and 9 are nearly aligned (cosine 0.967459), while every other candidate pair has cosine below 0.55. The filled-value vectors are strongly aligned: their 36 pairwise cosines range from 0.649 to 0.924, with mean 0.804. Filled values contribute little when attention concentrates on an empty cell; the environment supplies their constraints through the candidate masks. These measurements describe the information available through OV without establishing the necessity of individual components for the final action.

F.3.2 CANDIDATE SIMILARITY AND THE TEACHER'S DIGIT RULE

We enumerate all 502 candidate sets of sizes two through nine and apply the teacher's fixed digit-selection rule. None selects digit 9, and exchanging digits 8 and 9 leaves the selected digit unchanged in all 502 cases. This enumeration evaluates the digit-selection rule alone. It constructs no Sudoku states. The complete MRV trajectories contain no digit-9 targets among 1,636,562 forward decisions after constraint propagation. For these candidate sets, retaining the indicators for digits 1–7 and the sum $b_{i,8} + b_{i,9}$ is sufficient to recover the teacher's next-digit choice: a present digit from 1–7 takes priority; otherwise, the set must be $\{8, 9\}$ and the teacher chooses 8. Thus, sharing an additive candidate vector between 8 and 9 preserves the information required by this decision rule, although it does not preserve the full candidate set.

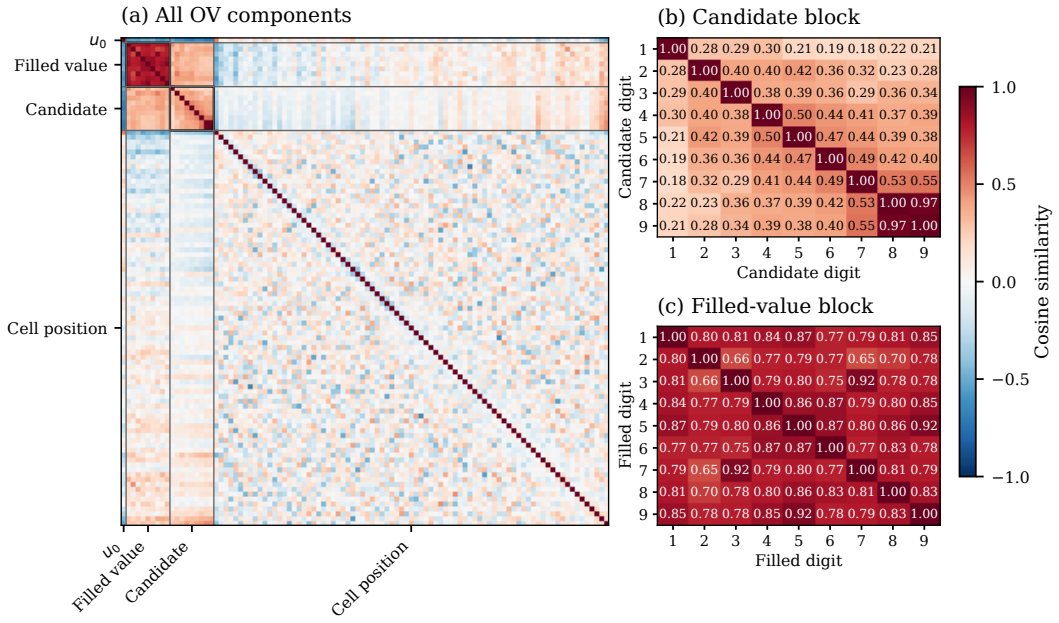


Figure 13: **Geometry of the OV output components.** **(a)** Pairwise cosine similarities among the baseline, filled-value, candidate, and cell-position vectors in (4). Position vectors are weakly aligned with content vectors on average. **(b)** Candidate-block zoom: digits 8 and 9 are nearly aligned (cosine 0.97). **(c)** Filled-value-block zoom: different filled digits are strongly aligned (mean cosine 0.80).

F.4 MLP FORWARD-ACTION SELECTION: REPRESENTATIONS AND CAUSAL TESTS

We first measure the MLP's contribution to the teacher-action margin. We then characterize its activation patterns, define activity-based neuron groups, and test how candidate projections affect digit choice. Each experiment specifies its state sample and intervention.

F.4.1 FORWARD ACTIONS AND MARGIN DECOMPOSITION

Action-margin decomposition. The primary sample contains 600 forward states from rollouts of 128 extreme puzzles for at most 80 steps, covering 77 selected-cell positions. In each eligible state, the model predicts a legal forward action, and the teacher-selected cell has at least two candidates. The second sample uses the 324 unchanged source states of the digit-permutation experiment in Appendix F.4.2, with four distinct puzzles per selected position and one per difficulty. These states are selected without conditioning on model outputs. The model matches the teacher's cell-digit action on every state in both samples. The samples serve complementary purposes: the first measures the MLP's contribution on states encountered during model rollouts, while the second checks whether the conclusion persists with equal coverage of all 81 selected-cell positions and all four difficulty levels, without filtering on model predictions. We report the rollout sample in the main text and retain this coverage check here.

The action margin (5) compares the teacher action a^* with the strongest alternative a^- among all other actions, including REVERT, filled-cell actions, and non-candidate digits. Let $\mathbf{w}_a = (W_{\text{act}})^\top_{a,:}$ denote the action-head vector for action a, $\mathbf{h}_s$ the residual stream entering the MLP, and $\mathbf{u}_s$ the MLP output. With the final RMS normalization factor α_s from the complete forward pass, the two contributions are

$$m_s^{\text{residual}} = \alpha_s(\mathbf{w}_{a^*} - \mathbf{w}_{a^-})^\top \mathbf{h}_s,$$
$$m_s^{\text{MLP}} = \alpha_s(\mathbf{w}_{a^*} - \mathbf{w}_{a^-})^\top \mathbf{u}_s.$$

The compared actions and normalization factor are held fixed across the two components. Reported shares divide the mean component contribution by the mean margin. Both contributions use activations from the unmodified model. Their sum differs from the action margin computed by the complete

forward pass by at most 0.091 logits across the 924 states because of finite-precision rounding. The MLP contribution exceeds the residual contribution on every state.

Sample	States	Action margin	Residual	MLP	MLP share
Forward rollout states	600	16.904	1.226	15.677	92.74%
Teacher-trajectory states	324	15.689	0.925	14.767	94.13%

Table 4: Contributions to the action margin. Margins and component contributions are means in logit units.

Replacing the 512 MLP activations with their mean over the 600-state sample reduces teacher-action agreement from 100% to 32.00%. Removing the MLP output gives 30.83% agreement on that sample and 20.37% on the 324-state sample. After each intervention, we recompute the final RMS normalization and all action logits.

F.4.2 MLP STATE CONSTRUCTION AND ACTIVATION COMPARISONS

Let $\mathbf{a}_s \in \mathbb{R}^{d_{\text{ff}}}$ be the 512 SwiGLU activations at the $\langle\text{EOS}\rangle$ token, before the MLP down projection. We compare two vectors using the relative distance

$$r(\mathbf{a}, \mathbf{a}') = \frac{\|\mathbf{a} - \mathbf{a}'\|_2}{\frac{1}{2}(\|\mathbf{a}\|_2 + \|\mathbf{a}'\|_2)}. \tag{8}$$

Smaller values indicate more similar activity. Top-ten overlap is the intersection of the ten largest-magnitude neuron-index sets, divided by ten; a fixed index ordering resolves ties. We first compare unchanged teacher states, then vary candidate identities through digit permutations.

Cell dependence on unchanged MRV states. We search the complete teacher trajectories of the $4{,}000$ puzzles in Appendix F.2.5 for states with the same selected candidate set at different positions. For each of $2{,}125$ eligible puzzles, we select three distinct states sharing the candidate set, the teacher's preferred digit, and the number of empty cells. States A and C select the same cell. State B selects a different cell. All grids and candidate sets are taken directly from the trajectories, including failed branches. Across these groups, the mean relative activation change is 1.240 for A-B and 0.00464 for A-C. The ten largest-magnitude activations share 15.50% and 99.73% of their neuron indices, respectively. The model matches the teacher's cell-digit action on all $6{,}375$ states. The state-selection criteria do not use model activations or predictions.

Digit permutations at fixed positions. For each of the 81 positions and each difficulty level, we uniformly select a puzzle containing a two-candidate MRV decision at that position, then uniformly select one such state. This gives 324 source states from 293 distinct source puzzles, with four distinct puzzles per position. State selection uses seed 20260914 and does not condition on the attention argmax or predicted action. All 324 source states are unchanged by constraint propagation.

A bijection of digits 1 through 9 is applied to the filled grid and all candidate sets. This preserves row, column, and box constraints, candidate counts, the teacher-selected cell with lowest-index tie-breaking, and whether constraint propagation changes the state. For each source state and each of the 36 possible candidate pairs at the selected cell, we uniformly sample 128 digit permutations that produce that pair, without replacement. We evaluate the resulting $324 \times 36 \times 128 = 1{,}492{,}992$ states with the unmodified model. The teacher's preferred digits are recomputed for the relabeled sets. Relabeling preserves state validity. The teacher's digit preference can give a different search path when solving a relabeled puzzle from its initial grid.

Activation comparisons and explained variance. For pairwise comparisons, we sample 32 of the 128 permutations for each source state and selected-cell candidate set, independently of model outputs. Within-set controls pair the first 16 with the remaining 16. Different-set comparisons pair corresponding samples for every pair of candidate sets. Cross-state comparisons pair corresponding samples from each pair of source states with the same selected cell and selected-cell candidate set. Means first average source states within each position and then weight all 81 positions equally.

Table 5 compares activations from individual Sudoku states, before forming the cell-candidate means used in Figure 14. At a fixed cell, different candidate sets give mean relative distances of 0.4732

when the preferred digit stays the same and 1.3219 when it differs. Different source states with the same selected cell and selected-cell candidate set give a smaller distance of 0.0589. The preference dependence seen in the group means therefore also appears in individual-state comparisons.

Comparison at a fixed cell	Pairs	Mean r	Top-ten overlap
Same set, same source state	186,624	0.0258	98.14%
Same set, different source states	559,872	0.0589	95.64%
Different sets, same preferred digit	870,912	0.4732	72.55%
Different sets, different preferred digits	5,660,928	1.3219	9.99%
All different-set pairs	6,531,840	1.2087	18.33%

Table 5: MLP activation comparisons under digit permutations. States retain the same teacher-selected cell. Pair counts include shared states and are not independent sample sizes.

Let $d_s^* = \min_{\prec} C_s$ denote the teacher's preferred digit in candidate set C_s, with selected cell i_s. For the evaluated state set $\mathcal{S}$, let $\boldsymbol{\mu}_{i,d}$ be the mean activation over states with selected cell i and preferred digit d, and $\boldsymbol{\mu}$ the overall mean. The fraction of total activation variance accounted for by these group means is

$$R^2_{\text{position, digit}} = 1 - \frac{\sum_{s\in\mathcal{S}} \|\mathbf{a}_s - \boldsymbol{\mu}_{i_s,d_s^*}\|_2^2}{\sum_{s\in\mathcal{S}} \|\mathbf{a}_s - \boldsymbol{\mu}\|_2^2} = 90.04\%. \tag{9}$$

This calculation uses all 128 sampled permutations per candidate set and source state, with equal weight per state. Each $\boldsymbol{\mu}_{i,d}$ averages all states at cell i whose candidate set has preferred digit d, pooling across the four source states and all such candidate sets. Preferred digits occur with different frequencies because they are selected from different numbers of pairs. Each cell has its own mean activation for each preferred digit. Grouping only by position explains 22.54% of the variance, and grouping only by preferred digit explains 18.68%, compared with 90.04% for their joint grouping.

For the full candidate-set comparison, each $\boldsymbol{\mu}_{i,C}$ averages four source states and their 128 permutations. Replacing $\boldsymbol{\mu}_{i_s,d_s^*}$ with $\boldsymbol{\mu}_{i_s,C_s}$ in (9) gives 99.28% explained variance. Its 0.72% residual includes variation across source states and relabelings at fixed (i, C). Computing the explained variance of candidate-set grouping separately within each position gives a median of 99.83% and a range from 90.67% to 100%. The minimum is at row 7, column 1. These group means and explained variances are descriptive fits on the evaluated states.

To assess transfer across source states, we estimate group means from three states at each position and predict activations on the fourth, repeating for all four held-out states. Pooling their squared errors and using the same total sum of squares as (9) gives $R^2 = 89.79\%$ for position and preferred digit, and 98.98% for position and the full candidate set. These quantities are computed from activation means and within-group sums of squares over the complete permutation sample.

Figure 14(a) plots $\frac{1}{81}\sum_i r(\boldsymbol{\mu}_{i,C}, \boldsymbol{\mu}_{i,C'})$. Distances are computed within each position before averaging across positions, preserving the cell dependence of the activation vectors.

Distances between cell-candidate blocks. Let $\boldsymbol{\mu}_{i,C}$ average activations for selected cell i and candidate set C over four source states and their 128 permutations. Define $D_{ij}(C, C') = r(\boldsymbol{\mu}_{i,C}, \boldsymbol{\mu}_{j,C'})$. We extend this calculation to every pair of the $81 \times 36 = 2{,}916$ activation means. The resulting $2{,}916 \times 2{,}916$ distance matrix consists of 81×81 blocks, each containing 36×36 candidate-set comparisons. For fixed (C, C'), Figure 14(b) plots $\frac{1}{6{,}480}\sum_{i\neq j} D_{ij}(C, C')$. The ordering includes both orientations of each of the $3{,}240$ unordered cell pairs.

Each mean uses the same four source states and 128 permutations per state as Figure 14. Constraint propagation leaves every source grid and candidate set unchanged, and no source state contains a singleton empty cell. Because different positions use different source-state cohorts, these comparisons describe representation differences without isolating a causal effect of position.

F.4.3 Individual-activation projections and neighborhood separation

For Figure 15, we select the first of the 32 permutations originally presampled for each source board and candidate pair. This gives $81 \times 4 \times 36 = 11{,}664$ individual activation vectors from the same 324

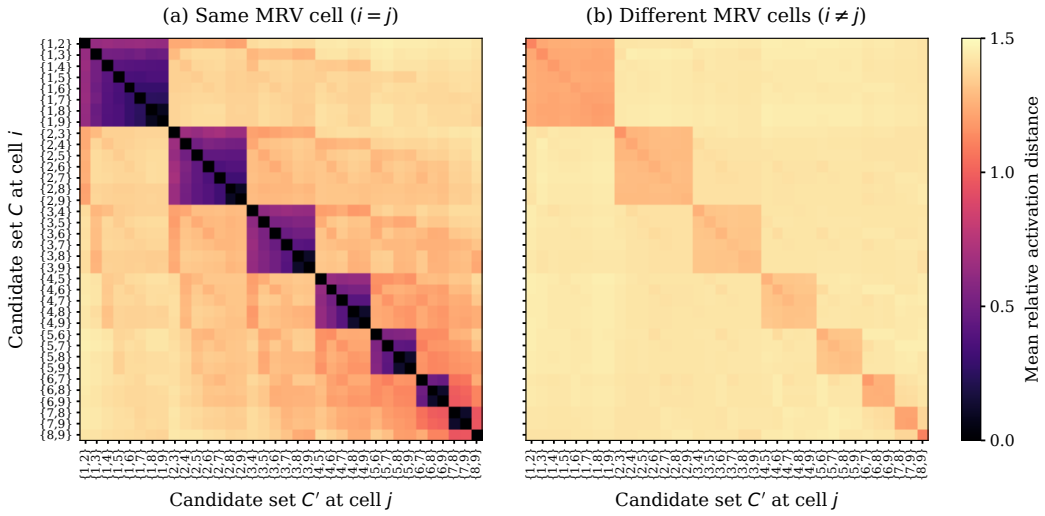


Figure 14: **MLP activations track the selected cell and preferred digit.** Panels compare mean activation vectors using (8). **(a) Same cell:** different candidate sets produce similar activations when the teacher chooses the same digit. **(b) Different cells:** activations differ even when the candidate sets are identical. Smaller digit labels indicate higher teacher preference.

Selected cells	Candidate sets	Mean	Min	Max	Std.
Same	Different	1.2078	0.0721	1.5497	0.3099
Different	Same	1.2421	0.8241	1.6012	0.1059
Different	Different	1.4064	0.8533	1.6163	0.0665
Different	All pairs	1.4018	0.8241	1.6163	0.0731

Table 6: Pooled distances between cell-candidate activation means. Rows summarize 102,060, 233,280, 8,164,800, and 8,398,080 ordered entries, respectively. Standard deviations describe the finite collection of distances and are not standard errors. Self-comparisons with the same cell and candidate set have distance zero.

source states and 293 distinct puzzles. Selection uses neither activation values nor model correctness. The sample retains its one model-action mismatch and five attention-cell mismatches. Labels denote the teacher-selected cell and preferred digit. Digit 9 never wins a two-candidate comparison, so eight preferred digits occur (Appendix F.3.2).

UMAP (McInnes et al., 2018) receives the raw 512-dimensional SwiGLU activation vectors, without standardization, vector normalization, or PCA preprocessing. We use Euclidean distance, 30 neighbors, minimum distance 0.1, 500 optimization epochs, spectral initialization, and random seed 20260925. No cell, digit, or action labels are supplied to the fit. Figure 15(a,b) uses separate fits with the same parameters, restricted to digit 1 and cell 0, respectively. These are the first labels in their respective orders, chosen without evaluating their separation. Holding one label fixed lets us inspect variation associated with the other: cell position in panel (a) and preferred digit in panel (b).

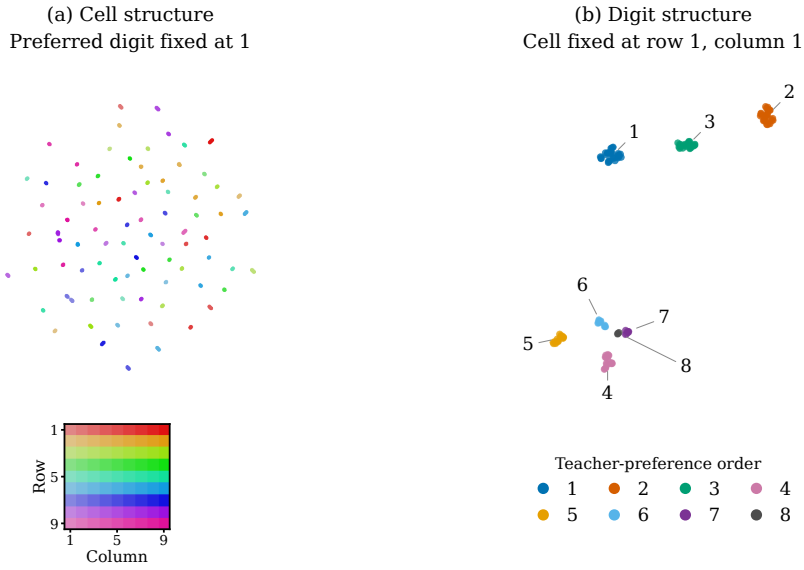


Figure 15: MLP activation structure by selected cell and preferred digit. Each point is one individual activation vector. **(a)** 2,592 inputs with preferred digit 1, colored by cell. **(b)** 144 inputs at row 1, column 1, colored by preferred digit. Each panel is a separate unsupervised UMAP fit; many points overlap within compact groups. Digit labels follow teacher-preference order.

We quantify separation in the original activation space by finding each vector's nearest Euclidean neighbor after excluding all vectors from the same source puzzle, including that puzzle's states at other cells. We also exclude the same candidate pair in a stricter comparison. The latter score omits the 324 queries with preferred digit 8: that digit wins only pair $\{8, 9\}$, so no different-pair reference can share its label. It evaluates the remaining 11,340 queries. The results are:

Reference restriction	Same cell	Same digit	Same joint action
Different source puzzle	99.974%	99.983%	99.957%
Different puzzle and candidate pair	99.974%	99.956%	99.938%

Giving each eligible joint action equal weight yields 99.968% and 99.952% joint agreement, respectively. For a complementary measure, let a_s be the mean distance from activation s to other activations with its joint cell–digit label, and let b_s be the smallest mean distance to any other joint-label group. The mean silhouette $(b_s - a_s)/\max(a_s, b_s)$ over all inputs is 0.630. These are descriptive label-based measurements; the 648 groups are not inferred by an unsupervised clustering algorithm. The observations share source puzzles and permutation families, so the percentages are not independent trial estimates or model action accuracies.

The original-space statistics above are independent of the UMAP projection. Figure 16 shows a linear comparison using the same observations. Its first two principal components explain only 11.20% of activation variance, so their overlap does not contradict separation in the full space.

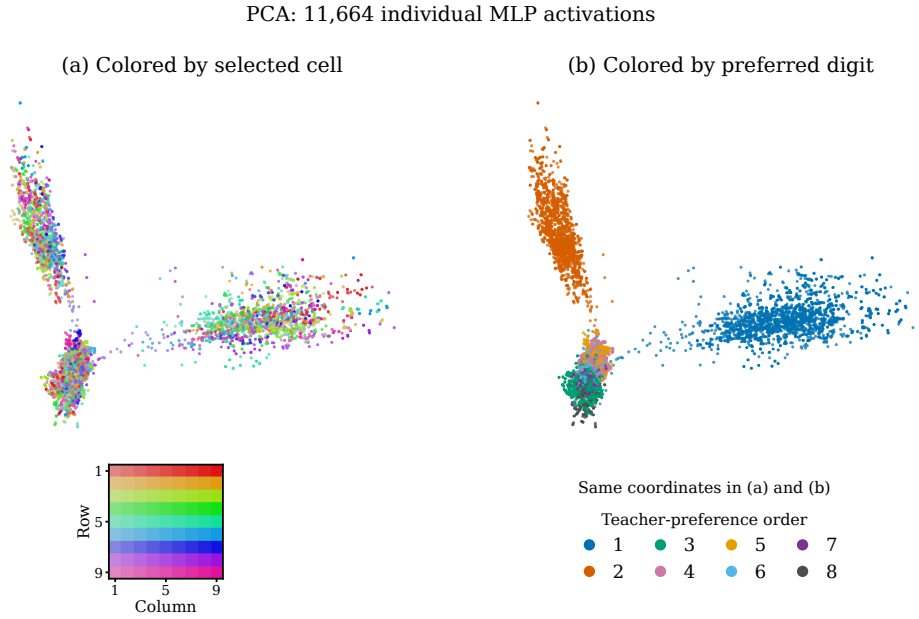


Figure 16: **Linear projection of the same individual MLP activations.** Both panels show the first two principal components of the $11{,}664$ raw activation vectors, colored by cell and preferred digit. The components explain 7.96% and 3.24% of variance, respectively. Colors and digit labels match Figure 15.

F.4.4 Prediction errors and singleton diagnostics

Prediction errors under digit permutations. The model matches $1{,}492{,}831$ of the $1{,}492{,}992$ teacher cell-digit actions. All 161 mismatches, spanning five positions, place a non-candidate digit at the teacher-selected cell. The attention argmax matches the teacher-selected cell on $1{,}492{,}582$ states (99.9725%). All outcomes remain in the activation analysis. For computational efficiency, the permutation calculation evaluates only the $\langle \text{EOS} \rangle$ query in the one-layer model. Complete model evaluations on $50{,}976$ relabeled states produce the same activation vectors, logits, and actions. We also verify all 552 states with an action or attention-selection mismatch.

Singleton diagnostic. Since permutations preserve candidate count, singleton sets use separate base states from intermediate states before singleton filling. We sample 207 such states across all 81 selected positions: one per position from each of the easy and medium sets, plus 36 hard and nine extreme examples. Each is a valid intermediate state generated during constraint propagation. All nine singleton sets receive 128 relabelings per base state, giving $238{,}464$ states. Attention selects the singleton cell throughout, but action agreement is 75.61%: $41{,}055$ states produce REVERT and $17{,}109$ produce a non-candidate digit at the singleton cell. The environment fills these singleton cells during constraint propagation, so these states do not occur at model decision steps.

F.4.5 Neuron selectivity and overlapping activity groups

We use all $1{,}492{,}992$ states for activation means and the same 32 sampled permutations per source state and pair ($373{,}248$ states) for per-neuron second moments. Let $e_{i,d,j} = \mathbb{E}[a_j^2 \mid i, d]$ average those squared activations over source states and candidate pairs with selected cell i and preferred digit d, eight observed preferences per cell. Define $G_{i,d} = \text{Top10}_j \, e_{i,d,j}$ as the ten neurons with largest mean squared activity for that cell–preference context. A fixed index ordering resolves ties. We use this single activity-based group definition throughout the projection and backtracking analyses.

There are $81 \times 8 = 648$ contexts and $d_{\text{ff}} = 512$ neurons, so these groups necessarily reuse neurons. Shared cell and digit features or superposition of action features in overlapping directions (Elhage et al., 2022) could both contribute; the overlap measurements do not distinguish them. Define a neuron's cell-activity share as

$$p_{i,j} = \frac{\frac{1}{8}\sum_{d=1}^{8} e_{i,d,j}}{\sum_{i'} \frac{1}{8}\sum_{d=1}^{8} e_{i',d,j}}.$$

The equal preference weights prevent digits that win more candidate pairs from dominating this selectivity measure. Squared activity counts both positive and negative SwiGLU responses. The strongest cell $\hat{\imath}_j = \arg\max_i p_{i,j}$ contributes a median share of 13.9%. Only 11 of 512 neurons exceed 50%, and none exceed 90%. The median effective number of cells, $1/\sum_i p_{i,j}^2$, is 17.0.

Within the strongest cell, the largest preference share $\max_d e_{\hat{\imath}_j,d,j} / \sum_{d'} e_{\hat{\imath}_j,d',j}$ has median 45.6%. In total, 209 neurons exceed 50%. Choosing the strongest cell and preference on three source states and evaluating on the fourth gives median shares of 13.8% and 45.7%. These held-out source-state comparisons are within each cell and do not evaluate generalization to new puzzles. Together, the results support mixed cell and preference tuning (Figure 17).

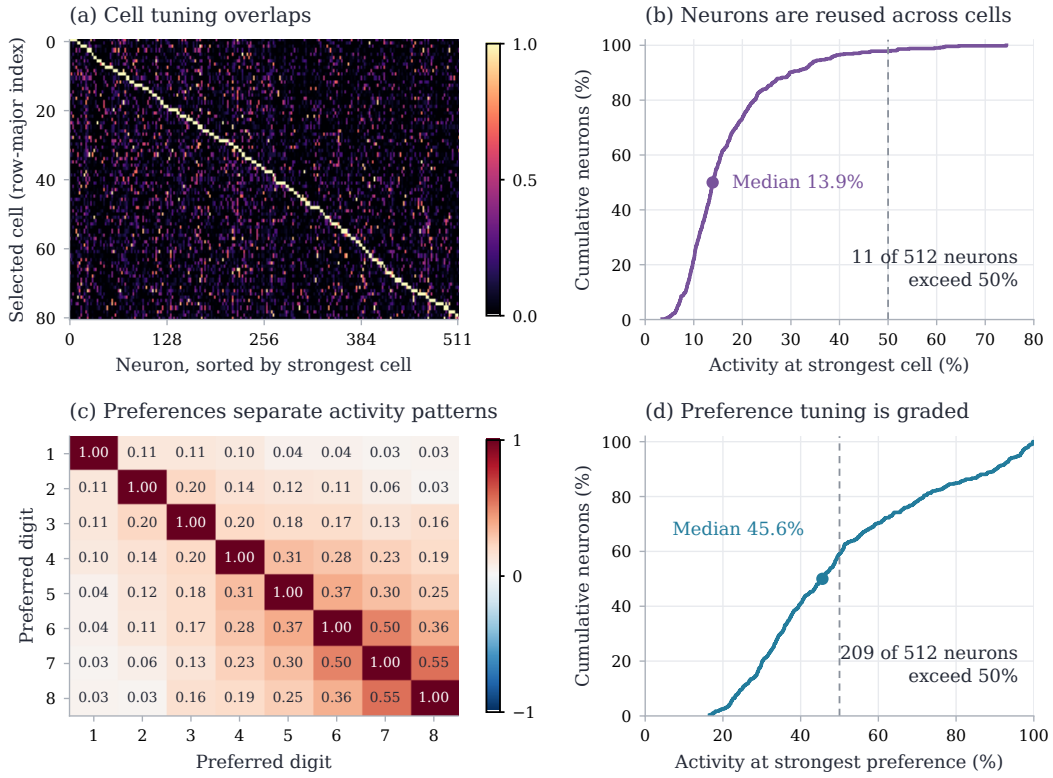


Figure 17: Neuron reuse and preference tuning. **(a)** Mean squared activity across cells, divided by each neuron's peak. Columns are sorted by strongest cell. The diagonal follows from this sorting, while off-diagonal activity shows reuse. **(b)** Cumulative distribution of the strongest cell's share of each neuron's total activity. **(c)** Cosines between cell-preference activation means, computed within each cell and then averaged over cells. **(d)** Cumulative distribution of the strongest preference's activity share at each neuron's strongest cell. Dashed lines mark 50% in (b,d). Preferences are weighted equally for these shares.

Geometry and index overlap provide complementary descriptions. Table 7 compares candidate-set means. Top-ten overlap is the intersection size divided by ten for their largest-magnitude neuron indices. The top ten capture 54.7% of the centroid squared norm on average, so this overlap summarizes only part of the activity. After pooling candidate sets by preference, different preference means within a cell have mean cosine 0.190 and 95th-percentile cosine 0.493. Only 31.5% of these pairs have absolute cosine below 0.1. Distinct activation patterns therefore do not establish universally orthogonal groups, nor does low cosine imply disjoint neuron support.

Candidate-set mean comparison	Mean cosine	Top-ten overlap
Same cell, same preference, different sets	0.881	73.2%
Same cell, different preferences	0.137	10.0%
Different cells, same set	0.228	14.7%

Table 7: Geometry and neuron-index overlap of activation means. Entries describe the finite evaluated collection. They are not estimates of independent puzzle-solving performance.

F.4.6 CANDIDATE PROJECTIONS AT THE NEURON LEVEL

For each neuron, we compute both $\gamma^g_{j,d} = (W_1)_{j,:}\mathbf{\Delta}_d$ and $\gamma^u_{j,d} = (W_2)_{j,:}\mathbf{\Delta}_d$ for all nine candidates, giving $512 \times 9 \times 2 = 9{,}216$ coefficients. The candidate direction uses the present-minus-absent embedding, because absence has a learned embedding of its own. Projections use the model's learned weights and the input scale $\lambda = 16/\sqrt{2}$.

Let $\mathbf{h}_{\text{fixed}}$ be the residual before attention at the $\langle\text{EOS}\rangle$ token. From (4), the selected-cell approximation to the MLP input residual is

$$\begin{aligned}\mathbf{h}(i, C) &\approx \mathbf{h}_{\text{fixed}} + \mathbf{u}_0 + \mathbf{u}_{\text{pos},i} + \sum_{d\in C} \mathbf{\Delta}_d,\\ \beta^g_{i,j} &= (W_1)_{j,:}(\mathbf{h}_{\text{fixed}} + \mathbf{u}_0 + \mathbf{u}_{\text{pos},i}),\\ \rho(i, C) &= \sqrt{\|\mathbf{h}(i, C)\|_2^2/d_{\text{model}} + 10^{-5}}.\end{aligned}$$

The up-branch baseline is defined analogously. The gate preactivation is therefore

$$g_j(i, C) \approx \frac{\beta^g_{i,j} + \sum_{d\in C} \gamma^g_{j,d}}{\rho(i, C)}. \tag{10}$$

The up preactivation has the analogous decomposition, and the resulting activation is $a_j = \text{SiLU}(g_j)u_j$.

How position affects activations and action scores. The candidate coefficients $\gamma^g_{j,d}$ and $\gamma^u_{j,d}$ are shared across positions. Cell identity enters through the baselines $\beta^g_{i,j}$ and $\beta^u_{i,j}$, and through the RMS factor $\rho(i, C)$. At a fixed candidate set C, changing the selected cell from i to i' changes the two baselines by

$$\begin{aligned}\beta^g_{i',j} - \beta^g_{i,j} &= (W_1)_{j,:}(\mathbf{u}_{\text{pos},i'} - \mathbf{u}_{\text{pos},i}),\\ \beta^u_{i',j} - \beta^u_{i,j} &= (W_2)_{j,:}(\mathbf{u}_{\text{pos},i'} - \mathbf{u}_{\text{pos},i}).\end{aligned}$$

The same candidate coefficients therefore combine with different baselines at different cells. The SwiGLU nonlinearity and multiplication make the resulting activation $a_j(i, C)$ depend jointly on position and candidates. Neurons can participate in multiple cell–digit groups, as quantified in Appendix F.4.5.

To connect these activations to cell identity in the output, let r denote an output cell and d an output digit. With ρ_{out} the final RMS normalization factor, the MLP contribution to this action's logit is

$$z^{\text{MLP}}_{(r,d)}(i, C) = \frac{1}{\rho_{\text{out}}} \sum_{j=1}^{d_{\text{ff}}} a_j(i, C)\mathbf{w}^\top_{(r,d)}(W_3)_{:,j}. \tag{11}$$

For a fixed digit, different output cells have different coefficients. Changing the selected position changes the activations supplied to these readouts, allowing the MLP to support the digit at the selected cell. The incoming residual also contributes to the final logits, as decomposed in Appendix F.4.1. The position comparisons in Appendix F.4.2 measure this dependence on observed states; they do not isolate a causal effect of the position vector alone.

Reconstruction and candidate-preference patterns. The reconstruction using (10) and its up-branch analogue explains 99.76% of the variance across the 2,916 cell-candidate activation means

obtained with unmodified attention. The mean cosine with the measured vectors is 0.9990. The variance denominator centers each neuron over these $2{,}916$ means. This is a weight-based reconstruction with no fitted parameters for activation means, distinct from the grouping statistic in (9) and from action accuracy.

We assign each neuron its strongest cell and, within that cell, its strongest preference using $e_{i,d,j}$, without inspecting candidate coefficient signs. Of all 512 neurons, 82.6% have a positive gate coefficient for their assigned preference. Across higher-priority coefficients of neurons assigned preferences from 2 to 8, 70.5% are negative. Among those eligible neurons, 42.4% have negative coefficients for every higher-priority candidate, and 30.0% satisfy both this condition and a positive own-candidate coefficient. Their mean higher-priority coefficient is -1.60, compared with $+0.81$ for lower-priority candidates, first averaging each bucket within a neuron and then across neurons. Thus the aggregate sign pattern does not hold for every neuron.

For Figure 18(a), we use the groups $G_{i,y}$ from Appendix F.4.5 and plot

$$M^g_{y,d} = \frac{1}{81}\sum_i \frac{1}{10}\sum_{j\in G_{i,y}} \gamma^g_{j,d}.$$

These groups may overlap across cells and preferences. Every diagonal entry of this averaged matrix is positive and every entry for a higher-priority candidate is negative. Lower-priority contributions are weaker on average but need not be negligible. Neuron 206 in panels (b,c) has the largest total mean squared activation among neurons assigned preference 3, without selection by coefficient signs. Its output weights also support this preference. At row 9, column 1 (cell index 72), the action-head coefficient $\mathbf{w}^\top_{(72,3)}(W_3)_{:,206}$ is 0.6191, while the coefficients for the other eight digits at that cell are all negative. After centering over the nine digit actions as in Appendix F.4.8, its coefficient is 0.6064, the largest among all 512 neurons for this action. These values use the effective bfloat16 weights with FP64 dot products. This output-weight check complements the activation-based selection of the example. Its row-9, column-1 gate baseline is $+6.25$; its gate coefficients are $+4.18$ for digit 3 and -11.29 for digit 1. Lower-priority candidate coefficients range from $+0.13$ to $+1.57$. The observed activations for $\{1,3\}$ and $\{2,3\}$ are -0.55 and 6.27, compared with values from 27.1 to 34.5 for pairs with another candidate of lower priority than digit 3. A negative gate does not force the SwiGLU activation to zero because SiLU is smooth and the up branch also contributes.

F.4.7 Held out comparisons and projection interventions

We use held-out source states to test whether the candidate projections identified above causally affect the tested digit margin. For every cell, each of its four source states is held out in turn. For a tested digit $y = 2, \ldots, 8$, choose $G_{i,y}$ using only the other three states. Comparing candidate pairs in which the teacher prefers the other candidate with pairs in which the teacher prefers y gives $2{,}268$ comparisons indexed by source state, cell, and digit. Mean group squared activity is lower when the teacher prefers the other candidate in 99.60% of comparisons. The median ratio of this activity to the activity when the teacher prefers y is 23.1%. These observations use count-matched relabeled states evaluated with unmodified attention, with sampling independent of model outputs.

The intervention compares adjacent pairs $\{y-1, y\}$ and $\{y, y+1\}$, for which the teacher prefers $y-1$ and y, respectively. One sampled permutation for each pair is chosen independently of model outputs. When the teacher prefers $y-1$, let α_i be the attention probability on the selected cell and ρ_s the pre-MLP RMS scale from the unmodified model. For $j \in G_{i,y}$, the gate correction is

$$g'_j = g_j + \frac{\alpha_i}{\rho_s}\left(\gamma^g_{j,y+1} - \gamma^g_{j,y-1}\right).$$

Up-only and joint corrections use the corresponding up coefficients. This is an internal intervention on the direct selected-cell contribution of $y-1$, not a replacement of the full counterfactual gate or an edit to the Sudoku input. Attention, the incoming residual, and all unselected neuron preactivations remain unchanged. SwiGLU, the down projection, final RMS normalization, and all 730 action logits are recomputed. The random-gate control applies the identical ten-component gate correction to ten randomly chosen neurons outside $G_{i,y}$, preserving the perturbation norm.

Define the tested-digit margin as $z_{(i,y)} - z_{(i,y-1)}$, distinct from the strongest-competitor action margin in (5). Its mean is -15.756 when the teacher prefers $y-1$ and 17.906 when the teacher prefers y. The



Figure 18: MLP neurons encode candidate preferences and respond to dead ends. **(a)–(c)** Three complementary views show how higher-priority candidates suppress neurons supporting lower-priority choices. **(a) Group coefficients:** averages within each neuron group and across 81 cells are positive for the preferred digit (boxed) and negative for higher-priority digits. **(b) Single-neuron coefficients:** for neuron 206, digit 1's negative gate coefficient outweighs digit 3's positive coefficient. **(c) Responses across candidate sets:** this neuron's activity falls when a higher-priority candidate accompanies its preferred digit 3. Predictions from weights (diamonds) closely match observed activations (bars); gray bars show pairs without digit 3.

(d) Backtracking. Clearing the selected cell's candidates reduces forward-action support (purple) and increases REVERT support (blue). Faint lines show 324 paired states; bold lines show means. See Appendix F.5 for the zero-candidate intervention details.

final action is (i, y) in 0% and 100% of cases, respectively. Table 8 reports the corrections. Gate-only correction increases this margin in 98.5% of cases and restores 76.2% of lost group squared activity, measured as the ratio of mean differences. Correcting both branches increases the tested-digit margin in every case, with a mean gain of 11.64 logits, and changes the final action to (i, y) in 21.7% of cases. The intervention shows that these candidate coefficients causally contribute to the tested-digit margin, without showing that the targeted group alone determines the model's action.

Correction when the teacher prefers $y-1$	Mean margin gain	Final action is (i, y)
None	0.00	0.00%
Gate	6.72	6.31%
Up	4.81	0.00%
Gate and up	11.64	21.74%
Random gate	0.29	0.00%

Table 8: Direct candidate-projection interventions on $2{,}268$ comparisons across held-out source states. Margin gains are in logit units and use the same baseline in which the teacher prefers $y-1$.

The $4{,}536$ baseline activation vectors agree exactly with the recorded values. Independent complete model evaluations reproduce all logits exactly for 32 baseline states and 16 states after the gate intervention. Groups selected on held-out source states test transfer within each cell. Cell comparisons still use different source-state cohorts and do not isolate a causal position effect.

F.4.8 Comparison with groups selected by action-head weights

For each cell-digit action, we identify ten neurons with the largest positive action-head coefficients for that action relative to the mean over the cell's nine digit actions. We call these its output-weight group. The group contribution sums their contributions to this relative action logit. Averaged across groups, gate weights align positively with the group's own candidate direction and negatively with higher-priority candidate directions. Explicitly, the relative-logit coefficient for neuron j at action (i, y) is

$$w_{i,y,j} = (W_3)_{:,j}^{\top} \left(\mathbf{w}_{(i,y)} - \frac{1}{9} \sum_{d=1}^{9} \mathbf{w}_{(i,d)} \right).$$

The ten largest positive coefficients select neurons capable of promoting the action when positively activated. In contrast, $G_{i,y}$ selects neurons by observed squared activity, including negative activations. Recomputing both definitions with the same learned weights and candidate scaling gives a mean overlap of 2.66 out of ten neurons across the 648 shared cell-preference contexts. Their 8×9 gate-coefficient matrices, averaged over cells, have Pearson correlation 0.962 across the 72 entries. Both group definitions therefore recover a similar aggregate coefficient pattern without identifying the same neurons.

Using 120 base states, we construct $8{,}640$ two-candidate states. At a fixed selected cell, the tested digit is paired with either a lower- or a higher-priority alternative. The model selects the correct action in both cases throughout. The mean relative-logit contribution of the output-weight group is 5.974 when the teacher prefers the tested digit and -0.011 when the teacher prefers the other candidate. For 840 matched pairs, we copy the tested group's gate and/or up values from the state in which the teacher prefers the tested digit to the matched state in which the teacher prefers the other candidate, keeping all other neurons unchanged. We measure the fraction restored by dividing the increase after the intervention by the difference between these two cases. The mean fraction is 60.8% for gate-only, 24.8% for up-only, and 94.0% for both. A gate replacement in ten control neurons gives 1.1%. Joint replacement changes the final action to the restored digit in 1.9% of cases.

F.5 Zero-candidate inputs and the Revert action

At a dead end, an empty cell has no candidates and the teacher selects REVERT; Section 3.1 calls such a cell a dead-end cell. Its candidate sum in (4) is zero, while the shared baseline still contains the absent-candidate embeddings. We test how this input changes the MLP's support for forward actions and REVERT, and then ablate the leading contributors.

F.5.1 Observed rollouts and competing-cell edits

Rollouts cover 96 extreme puzzles for at most 120 steps. Among 11,412 decisions, all 5,344 dead-end states produce REVERT and all 6,068 forward states produce forward actions. Separate candidate-set interventions start from 400 forward states. They remove candidates at the teacher-selected cell or the least-constrained other empty cell, holding the grid and all other candidate sets fixed. Clearing the candidate set yields REVERT in $400/400$ cases at either location. One-candidate controls yield false REVERT in $73/400$ selected-cell edits and $8/400$ competitor edits.

F.5.2 Selected-cell edits and forward-action support

We use the 324 two-candidate source states of Appendix F.4.2, with one state for each of 81 selected positions in each of the four difficulty splits. For each state, we clear the teacher-selected cell's candidate set, keeping the grid and all other candidate sets fixed, and evaluate the model before and after the edit. Before the edit, the model selects the teacher action in every case. All edited states select REVERT. For the original action (i, d), we retain its previously defined top-ten activity group $G_{i,d}$ from Appendix F.4.5. Groups are not reselected after the edit. For a neuron group S and action a, the group's logit contribution is

$$c_{S,a}(s) = \frac{1}{\rho_{\text{out}}(s)} \sum_{j \in S} a_j(s)\, \mathbf{w}_a^\top (W_3)_{:,j},$$

where ρ_{out} is the RMS normalizer after the MLP residual addition. For the forward-action comparison, $S = G_{i,d}$. We recompute this normalizer for each member of the pair and compare contributions to the same original forward-action logit. The contribution decreases in $324/324$ pairs, from a mean of 11.924 to 1.068 logits. The group's summed squared activation, $\sum_{j \in S} a_j^2$, decreases in $321/324$ pairs. The median ratio of the edited to original summed squared activation is 0.0894.

F.5.3 Input projections of leading Revert contributors

Minimum-candidate attention routes an empty cell with no candidates to the ⟨EOS⟩ token. Under the selected-cell approximation, the candidate sum in (10) then vanishes, leaving $g_j(i, \varnothing) \approx \beta^g_{i,j}/\rho(i, \varnothing)$ and $u_j(i, \varnothing) \approx \beta^u_{i,j}/\rho(i, \varnothing)$. We retain neuron identities from the separate 400-pair ranking in Appendix F.5.4, based on the contribution increase after clearing the competitor cell's candidate set. Both gate and up coefficients use the same OV candidate directions and learned weights as (10). The coefficients for neuron 406 are shown below.

Candidate d	1	2	3	4	5	6	7	8	9
$\gamma^g_{406,d}$	−5.35	−6.65	−6.48	−7.87	−7.61	−7.92	−8.39	−6.58	−6.74
$\gamma^u_{406,d}$	9.35	11.50	7.42	6.12	8.80	6.42	8.20	5.62	5.83

Across positions, $\beta^g_{i,406}$ ranges from 5.60 to 31.28 and $\beta^u_{i,406}$ from −26.23 to −2.35. Thus, with no candidates, its positive gate and negative up value produce a negative activation. Every present candidate opposes both baseline projections. Its output coefficient $\mathbf{w}_{729}^\top (W_3)_{:,406}$ is −0.980, so the negative activation contributes positively to REVERT. In the paired model evaluations, its mean gate changes from 0.242 to 4.744, its mean up value from 0.284 to −4.570, and its mean activation from 0.065 to −21.388. Its mean contribution to the REVERT logit changes from −0.019 to 6.311. Neuron 339 also has nine negative candidate gate coefficients (−10.52 to −5.83). Its mean activation changes from −0.465 to −23.634, and its mean REVERT contribution from 0.173 to 9.869. Other contributors need not share these signs: neuron 348 has a positive output coefficient (0.849) and mean zero-candidate-state activation 9.169. The blue curve in Figure 18(d) sums the contributions of neurons 339 and 406 to the REVERT logit on the same 324 paired states. These two neurons are the top contributors in the separate 400-pair ranking. They are not reselected on the plotted states. Their summed contribution increases in every pair, from a mean of 0.154 to 16.181 logits. The two plotted groups have different sizes and action-head coefficients: ten forward-action neurons versus two REVERT contributors. The curves show group contributions rather than full action logits or margins, and do not assume exclusive neuron roles.

Using the baseline and positional contributions, the gate and up projections, and pre-MLP RMS normalization, the selected-cell approximation predicts the activations after these edits with mean

absolute error 0.0097 across all 324×512 entries. For neurons 339 and 406, the mean absolute errors are 0.098 and 0.086, respectively. This supports the accuracy of the selected-cell approximation on these edits. The comparison is descriptive and does not establish held-out neuron selection or a universally exact zero-candidate detector.

F.5.4 CONTRIBUTION RANKING AND ABLATIONS

The MLP decomposition uses 400 matched pairs of unedited and zero-candidate states, with the competitor's candidate set cleared. The MLP accounts for approximately 27.2 of the 31.5-logit mean increase in the REVERT logit. The top 2, 9, and 20 neurons account for approximately 50%, 80%, and 90% of this MLP contribution, respectively. Neurons are ranked by their mean contribution to the increase in the REVERT logit after the edit and ablated on the same 400 pairs. The top-20 margin reduction is therefore a descriptive same-sample intervention, not a held-out test of neuron ranking. Controls use a deterministic set of 20 other neuron indices. The mean REVERT margins, $z_{729} - \max_{a \neq 729} z_a$, are 22.78 intact, -1.21 after top-20 ablation, and 23.38 under the control. Their REVERT rates are 100%, 35%, and 100%. Ablating all 512 activations leaves a mean margin of 6.04 and a 97.25% REVERT rate. Larger ablations can affect both competing logits and normalization, so margin changes need not be monotone. These measurements identify an important subset of neurons. Since REVERT is still selected in 97.25% of pairs with all 512 activations ablated, the MLP as a whole is not necessary for REVERT on these states. The contribution reconstruction differs from the REVERT logit from the complete forward pass by at most 0.145 logits on the zero-candidate states, so the reported logit shares are approximate numerical decompositions.