# Continual Learning and Out of Distribution Generalization in a Systematic Reasoning Task

**Mustafa Abdool**
Department of Computer Science
Stanford University
moose878@gmail.com

**Andrew J. Nam**
Department of Psychology
Stanford University
andrewnam@stanford.edu

**James L. McClelland**
Department of Psychology
Stanford University
jlmcc@stanford.edu

## Abstract

Humans often learn new problem solving strategies from a narrow range of examples and generalize to examples out of the distribution (OOD) used in learning, but such generalization remains a challenge for neural networks. This impacts learning mathematical techniques, which can apply to unbounded problem spaces (e.g. all real numbers). We explore this limitation using neural networks trained on strategies for solving specified cells in $6 \times 6$ Sudoku puzzles using a novel curriculum, where models first learn two preliminary tasks, then we assess OOD generalization during training on a subset of the set of possible training examples of a more complex solution strategy. Baseline models master the training distribution, but fail to generalize OOD. However, we introduce a combination of extensions that is sufficient to support highly accurate and reliable OOD generalization. These results suggest directions for improving the robustness of models trained with the highly imbalanced data distributions in natural data sets.

## 1 Introduction and Related Work

The ability to learn new skills and infer abstract rules from limited data is a highly desirable property for machine learning systems, especially in the domain of mathematical problem solving, in which highly generalizable principles and problem solving strategies abound. Large language models (LLMs) with billions of parameters may be able to support many advanced cognitive abilities if their training data samples these abilities widely enough, but their ability to learn truly novel information and to exhibit out of distribution generalization (OODG) remains limited [1].

Here we address this limitation through the popular puzzle game Sudoku, which affords this opportunity due to the inherently abstract nature of the rules and reasoning strategies required. Although Sudoku has been solved using a neural network model [2], the network used for this fails to generalize OOD [3], as often happens with neural networks. In contrast, humans participants who learn a multi-step Sudoku strategy (the *Hidden Single* strategy) from a narrow range of training examples show strong OODG. In earlier work, we found that small-scale transformers could generalize the strategy OOD [3], but this required continuously interleaving the narrowly sampled Hidden Single (HS) examples with examples of component strategies spanning the entire distribution of puzzle instances. Such an approach is data-inefficient, and it is a limitation compared to the human participants in [4] who learned and generalized without interleaving.

In this work, we achieve reliably accurate OODG of the Hidden Single technique without interleaving simpler strategies. We define *reliably accurate OODG* as achieving 90% correct final accuracy on OOD problems in 10 of 10 model runs each with a separate random seed. We describe several techniques that significantly improve OODG over the baseline model previously used in [3]. Importantly, our baseline model is able to solve unseen examples when trained on a dataset drawn from the full distribution of puzzles. However, it fails to generalize OOD when trained with a restricted subset of

37th Conference on Neural Information Processing Systems (NeurIPS 2023) Workshop on MATH-AI.

puzzles for the HS strategy, even after previously learning from the full distribution of two component strategies. Thus, the baseline model has sufficient capacity to solve the full problem but exhibits poor OODG in this regime. We then introduce several extensions that allow successful OODG in this curriculum learning setting. When combined, these extensions support reliably accurate OODG on a challenging restriction of the training data distribution, as we detail below. Our contributions are:

1. We focus attention on out-of-distribution generalization in a cumulative learning setting where new skills build on abilities previously acquired.
2. We introduce a task setting and dataset in which a generic baseline model fails to exhibit OODG, despite having sufficient capacity to solve the task when trained on the entire distribution of problems.
3. We identify several factors that each contribute to achieving robust OODG over the baseline model.

## 2 Methods

### 2.1 Problem Description

**Problem overview.** Our approach is inspired by the approach taken in a tutorial used with human participants [3]. We train networks to evaluate whether a specified digit is or is not the correct solution to a specified target cell in a specified house type (row, column or $2 \times 3$ box) in a given $6 \times 6$ Sudoku grid. A valid Sudoku solution is one in which each unique house contains each digit exactly once. By specifying the target digit, cell, and house type in a partially completed grid, as in the human experiment in [3], we avoid the challenge of finding good target cells, house-types, and digits in Sudoku puzzles, focusing on OODG of the process of evaluating a possible solution once a possible target cell, solution digit, and house-type have been specified.

The network is given a board state along with a target cell and target digit that might or might not go in that cell (see Table 1.) A house type specification (row, column, or box) is also given for two of the three tasks (*Full House* and *Hidden Single*). The final output in these cases specifies whether or not the digit must go in the cell. Note that the solution is only 'yes' if there is sufficient information to determine that the target digit can go in the specified cell with certainty.

**Sudoku Tasks.** The model training data consists of examples from three tasks.

1. *Full House*: Determine whether every cell other than the target cell in the specified house is filled with a digit other than the target digit. If yes for all cells in the house other than the target cell, the target digit must go in the target cell.
2. *Can Contain*: Determine whether a target cell can contain the target digit. The cell must be empty and the target digit must not already be present in the same row, column, or box. Note that each instance of the Can Contain task queries 5 cells, as in the other two tasks, to balance the number of cells trained across tasks.
3. *Hidden Single*: Iterate over the non-target cells in the specified house (as in the Full House strategy) and check whether any of these cells can contain the target digit. If none of the other cells can contain it, then the target digit must be placed in the target cell.

The *Hidden Single* task requires combining the procedures required for the other two tasks, thus making it a good target for investigating compositional reasoning through curriculum-based learning.

**Training curriculum and assessment of OODG.** All our experiments are conducted in a *curriculum learning* setting where the model is trained sequentially on the Full House, Can Contain, and Hidden Single tasks (See SM Figure 2), and all involve 10 model runs with different random seeds. To solve the Hidden Single task, the model must combine the skill of iterating over cells in a certain house (from the Full House dataset) and checking for direct constraints (from the Can Contain dataset). The Full House and Can Contain training data was sampled from the *full distribution* of possible puzzles (in terms of *digits* and *house types*), so that the network has the opportunity to learn to encode and decode input and output tokens and develop a general solution for these constituent strategies. However, the Hidden Single task is trained on a restricted set of puzzles, where key dimensions such as *digits* or *house types* are held out. As it learns the Hidden Single task, we test the network on held out puzzles to assess how well it generalizes OOD.

Table 1: An input grid with example token sequences for all three tasks. Prompt text in normal font, target / model-generated tokens in bold. Note that, unlike this example, an independent grid is generated for each instance of each task.

| | | | | | |
|---|---|---|---|---|---|
| | 5 | | 6 | 2 | |
| 2 | | 6 | 1 | 3 | 5 |
| | 6 | | 2 | 5 | 1 |
| 5 | | | 3 | | 6 |
| 6 | | | 5 | 1 | 3 |
| | | 5 | | 6 | 2 |

| Full House | Can Contain | Hidden Single |
|---|---|---|
| <SOS>full_house | <SOS> | <SOS>hidden_single |
| goal_cell row 2 column 2 | digit 2 | goal_cell row 6 column 2 |
| house_type box | can_contain | house_type column |
| digit 4 | row 4 column 3 **yes** | digit 3 |
| is_filled | row 1 column 1 **no** | can_contain |
| **row 1 column 1 no** | row 2 column 3 **no** | **row 1 column 2 no** |
| **row 1 column 2 yes** | row 5 column 2 **yes** | **row 2 column 2 no** |
| **row 1 column 3 no** | row 4 column 1 **no** | **row 3 column 2 no** |
| **row 2 column 1 yes** | <EOS> | **row 4 column 2 no** |
| **row 2 column 3 yes** | | **row 5 column 2 no** |
| **solution no 4 <EOS>** | | **solution yes 3 <EOS>** |

### 2.2 Model Description

**Baseline transformer architecture and task sequence conventions.** We use a 3-layer transformer with a dimensionality of 90 as our baseline model (SM Figure 1). The model receives as input the Sudoku grid, followed by and a sequence of prompt tokens. We encode the grid as a flattened sequence of individual cell encodings, where each cell is represented by concatenating 30 dimensional learned embeddings of the cell's row $r$, column $c$, and contents (digit or blank) $d$ into a 90 dimensional embedding vector. The sequence of text tokens (the *prompt*) are added to position tokens and embedded into a 90 dimensional vector.

The model output is also a sequence of text tokens, sampled autoregressively. For the Full House and Hidden Single problems, the output sequence contains an iteration over all cells in the house that the target cell belongs to of the specified type, along with the answer to the "sub-problem" for each iterated cell. Next comes a 'solution' token, then the answer ('yes' or 'no') indicating whether the target digit must go in the target cell, then a repetition of the target digit. For the Can Contain task, the prompt specifies a target digit and the 'can-contain' token, then iterates over 5 randomly-chosen target cells, with a single output token ('yes' or 'no') for each, as shown in Table 1.

**Baseline model performance.** We first train the baseline model on examples drawn from the full distribution of HS puzzles to verify that our model has sufficient capacity to learn the hidden single technique. We then test for OODG by training the same architecture on increasingly restrictive puzzle distributions by holding out digits only and then both digits and one house type. Even in the least restrictive cases, we find that the baseline model only reaches 30.6% OODG accuracy despite solving unseen within-distribution problems with near 100% accuracy.

### 2.3 Extensions that Enhance OODG

**Larger transformer dimension with and without weight sharing.** Although the baseline model learns to perform the HS task when trained with samples from the full distribution of HS problems, it fails to generalize OOD with even two held out digits. For our first extension to the model, we increase the transformer dimension (TD) from 90 to 252. We explore two variants of this, one of which controls for the number of learnable parameters by sharing the weights across transformer layers as in [5, 6]. In the other, non weight-sharing variant, the parameter count is about 2.5 times that of the baseline model.

**Shared digit representation for board and text.** In the baseline model, the digits on the board are embedded with the same matrix as the row and column indices, while the digits in the prompt text (e.g., the target digit) are embedded using different weight matrices as in [3]. Since board and text digits correspond, our second extension uses the same embedding matrix for digits on the board and in the text while using a different embedding matrix for row and column indices. We hypothesize that this might promote alignment and improve OODG.

**Separate board architecture.** In the baseline model, the board (after flattening) is concatenated with the text tokens to form the input layer for the transformer stack and shared weights are used for processing both the board state and the solution sequence, possibly causing unhelpful competition. To address this possibility, our third extension keeps the baseline architecture, which becomes a solution sequence decoder, and uses a separate board encoder network to process the board state for use by the solution sequence decoder. The board encoder network is stripped down in that it only maps each of the cell embeddings to a key and a value with its own $W_k$ and $W_v$ matrices. The single set of keys and values is queried by each of the three transformer layers in the solution-sequence decoder.

**Encouraging OODG using synaptic intelligence.** In any curriculum learning setting, new learning can degrade connection-based knowledge acquired in previous tasks. For example, when training on only 2 out of 3 house types in the Hidden Singles task, weights supporting iteration over the held out house type that are acquired in the Full House task might become degraded, interfering with OODG to problems using the held-out house type. To mitigate this, we experiment with synaptic intelligence (SI) [7] which discourages changes to parameters that are considered important in minimizing the loss in previous tasks. Note that we use SI to explore whether it can enhance OODG rather than to avoid interference with performance on previous tasks as is more commonly done [8]. As a comparison to SI, we also train models where we decrease the learning rate (labeled LR Decay in Tables) at each task boundary in the curriculum.

**Performance measures.** The results tables below show results obtained with 10 runs of each model variant. *Peak OOD Acc* refers to the highest accuracy achieved on the test set of OOD examples as the network learns on the in-distribution training examples. *Final OOD Acc* refers to accuracy on the same problems at the end of the HS training phase (see SM for further details).

## 3 Results

**Digit generalization.** We first perform a comprehensive study to understand the impact of our extensions on OODG relative to the baseline model (See Table 2), using a training set in which 2 of the 6 digits are held out from use as the target digit. The baseline model with only LR decay performs quite poorly (top line in Table 2), and the larger transformer dimension did little by itself. However, weight sharing, shared digit representation, and separate board architecture all improve peak OODG accuracy, up to 96.5% accuracy when combined (bolded row before SI in Table 2). In all these cases, generalization performance erodes with continued training. Interestingly, SI alone seems to over-constrain the baseline model, preventing it from even being able to solve within-distribution (WD) problems from the training set. However, the impact of SI significantly improves when increasing the embedding and transformer dimension from 90 to 252, reaching over 90% accuracy on almost all runs with or without weight sharing. We next examine a more challenging OOD setting, holding out 4 of the 6 digits as targets during HS training (SM Table 6). Here, SI improves OODG with or without weight sharing. The representation and architecture extensions are not beneficial without SI, but combining them with SI allows the model to achieve reliably accurate OOGD.

Table 2: Results for Two Digit OODG

| Larger TD | Weight Sharing | Rep & Arch | LR Decay | SI | Median WD Acc | Median Peak OOD Acc | Mean Final OOD Acc | Median Final OOD Acc | Runs over 90% OOD Acc | Mean updates to 90% OOD Acc |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | ✓ | | 100% | 58.7% | 39.7% | 37.8% | 0% | - |
| ✓ | | | ✓ | | 100% | 59.1% | 41.3% | 40.0% | 0% | - |
| ✓ | ✓ | | ✓ | | 100% | 81.1% | 74.7% | 71.7% | 40% | 320 |
| ✓ | ✓ | Digits | ✓ | | 100% | 85.4% | 55.5% | 59.0% | 40% | 620 |
| ✓ | ✓ | Board | ✓ | | 100% | 88.2% | 66.5% | 65.2% | 40% | 524 |
| ✓ | ✓ | Both | ✓ | | 100% | **96.5%** | **71.8%** | **72.6%** | **80%** | 452 |
| | | | | ✓ | 55.2% | 55.2% | 56.1% | 53.1% | 10% | 1628 |
| ✓ | ✓ | | | ✓ | 100% | 98.9% | 88.1% | 95.1 | 80% | **116** |
| ✓ | | | | ✓ | 100% | **100%** | **90.1%** | **98.2%** | **90%** | 152 |

**House type and digit generalization.** Our final experiments further restrict the training set by holding out an entire target house type *(row)* as well as 4 of the 6 digits as targets. Holding out all rows makes OODG especially challenging, since the model can only succeed by combining what it learned about iterating over rows in the Full House phase with its learning about the full house task from training with the column and box house types. As shown in Table 3, the 252-dimension model with the representation and architecture extensions, SI, and without weight sharing achieved reliably accurate OODG, and did so after an average of only 104 gradient updates. Other models usually reached high OODG accuracy within 200-1000 gradient updates, but this gradually declined as the models continued to train on the restricted HS training set. For more detailed results, see SM Table 7 and SM Figure 4.

Table 3: Results for four-digit and house type OODG

| Larger TD. | Weight Sharing | Rep & Arch | LR Decay | SI | Median Peak OOD Acc | Mean Final OOD Acc | Median Final OOD Acc | Runs over 90% OOD Acc | Mean updates to 90% OOD Acc |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | | | ✓ | | 60.5% | 35.7% | 36.0% | 0% | - |
| ✓ | | | | ✓ | 98.2% | 80.6% | 93.6% | 60% | 192 |
| ✓ | ✓ | | | ✓ | 87.9% | 74.3% | 77.0% | 20% | 1046 |
| ✓ | | Both | | ✓ | **100%** | **95.7%** | **98.7%** | **100%** | **104** |
| ✓ | ✓ | Both | | ✓ | 90.2% | 77.0% | 82.1% | 50% | 926 |

## 4 Open Questions and Future Directions

Today's neural networks often appear to learn new tasks from just a few examples or instructions. However, they can be extremely sensitive to prompting details and often fail to generalize knowledge tuned into their weights [1]. Our work contributes to addressing this problem. Beginning with a transformer network that could learn our target Sudoku task but could not generalize to OOD examples, we find extensions that overcome this limitation within our specific problem domain. However, we do not see our work as providing a general solution. We hope the comments below indicate some of the open issues and approaches that could be pursued to address them.

**Avoiding problem-specific inductive biases:** Our representation and architecture enhancements prove important in our specific task and architecture context, but they can seem problem specific. Separating spatial and sequential networks while sharing representations of corresponding elements (e.g. printed and spoken words) may be broadly useful, but as used here, they seem to tailor the architecture to the problem. This points toward a broader issue that may be more generally relevant: While problem-specific inductive biases can surely be useful, we believe it is important to find solutions that work in large-scale models that learn to solve a wide range of different tasks.

**Finding general solutions for learning in natural contexts:** The various combinations of our extensions and restrictions of the range of data produce a complex pattern of effects on OOD generalization. This poses a significant challenge for the development of models that can generalize well on a wide range of tasks, since these inter-dependencies are likely to vary from task to task.

**Achieving human-level data efficiency:** Even when we achieve reliably accurate OODG from a narrow range of examples in our final experiment, this still requires over 100 gradient updates based on about 20,000 training examples. In contrast, humans can typically learn the hidden single strategy in fewer than 10 examples [4].

One approach to tackling these issues may rely on in-context learning [9] rather than the in-weights learning we explore in this paper. In-context learning may be less narrowly focused [1], requires only a few examples, and may prove to be more independent of architectural details. One possibility is that a general solution will involve initial in-context learning, with the results stored for later re-use in a hippocampus-like fast learning system, followed by gradual consolidation in weights [10].

## References


[1] Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. The reversal curse: Llms trained on "a is b" fail to learn "b is a", 2023.

[2] Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks, 2018.

[3] Andrew J. Nam, Mustafa Abdool, Trevor Maxfield, and James L. McClelland. Achieving and understanding out-of-distribution generalization in systematic reasoning in small-scale transformers, 2022.

[4] Andrew J. Nam and James L. McClelland. Systematic human learning and generalization from a brief tutorial with explanatory feedback, 2023.

[5] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers, 2019.

[6] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[7] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence, 2017.

[8] Andrea Cossu, Antonio Carta, and Davide Bacciu. Continual learning with gated incremental memories for sequential data processing. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2020.

[9] Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, Aaditya Singh, Pierre Richemond, James McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 35:18878–18891, 2022.

[10] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[11] Yuxuan Li and James L. McClelland. Systematic generalization and emergent structures in transformers trained on structured tasks, 2022.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

# 5 Supplementary Material

## 5.1 Overview of rules and terminology.

In Sudoku, the goal is to fill out a NxN grid so that exactly one of each digit from 1-N appears in every *house* where a house refers to a particular instance of a *row*, *column* or *box*. In addition, every cell on the board should be contained in exactly one type of each house. Although Sudoku is typically played on a 9x9 grid, we use 6x6 grids (with 2x3 boxes) which require similar complex reasoning procedures while reducing computational requirements.

## 5.2 Training Data Details

The *positive* examples for each task (Full House, Can Contain and Hidden Single) were obtained by incrementally solving valid 6x6 Sudoku puzzles (ie. puzzles with a single unique solution) and checking if the strategy was applicable to solve any cell during the process.

Generating the negative examples for each of the corresponding tasks was slightly more complicated as just sampling random incorrect digits for a given cell would likely be too easy. Specifically, for the *Hidden Single* strategy, we want the model to learn how to combine the answers for the *Can Contain* sub-problems rather than focusing on just eliminating a digit due to direct constraints. This idea is similar to studies of humans learning to play Sudoku as in [4] - where a *distractor* digit was used as an example of a more plausible negative choice. Table 4 below describes the negative sampling strategies for each of the three tasks. All three task datasets were *balanced* to have an equal number of positive and negative instances.

Table 4: Negative sampling strategies for each of the various task types

| Task Type | Negative Sampling Strategy |
|---|---|
| Full House | Sample from houses where there are unfilled cells other than the target cell or one of the cells contains the target digit |
| Can Contain | Sample from cells and target digits where the target digit has at least one direct conflict |
| Hidden Single | Sample from cells, target digits and houses where the target digit has *no* direct constraints but the Hidden Single strategy is not valid to solve that cell (ie. the digit can be placed in another cell in the house) |

## 5.3 Baseline Model Architecture Details

A diagram of the baseline model architecture is shown in Figure 1.

## 5.4 Model Details

The core of the baseline model is a 3-layer transformer encoder supported by an embedding weight matrix for digits, grid cell positions (row and column) and the input text. To create the final representation for each board cell, the embedding for the digit, row and column are concatenated together. There is also a decoder layer to output a final distribution over tokens in our small vocabulary (around 30 tokens total). Note that while all tokens after the prompt starts use the typical autoregressive attention mask (ie. they are unable to attend to future tokens), the initial tokens which make up the board itself (after flattening it into cells) use a special attention mask that allows any board token to attend to any other board token. In contrast, the tokens in the prompt can only attend to previous prompt tokens as in the usual transformer decoder case.

The overall dimension of the transformer inputs and outputs are 252 or 90 (in the *Larger TD* case or default case, respectively). All text tokens are encoded using an embedding with the same dimensionality as the transformer layer and position information is added to the vectors using the *Sorted Random Label* strategy from [11]. The 36 grid cell vectors and all token vectors are passed to the transformer, which is composed of 3 encoder layers with 6 heads and 1024-dim feed-forward

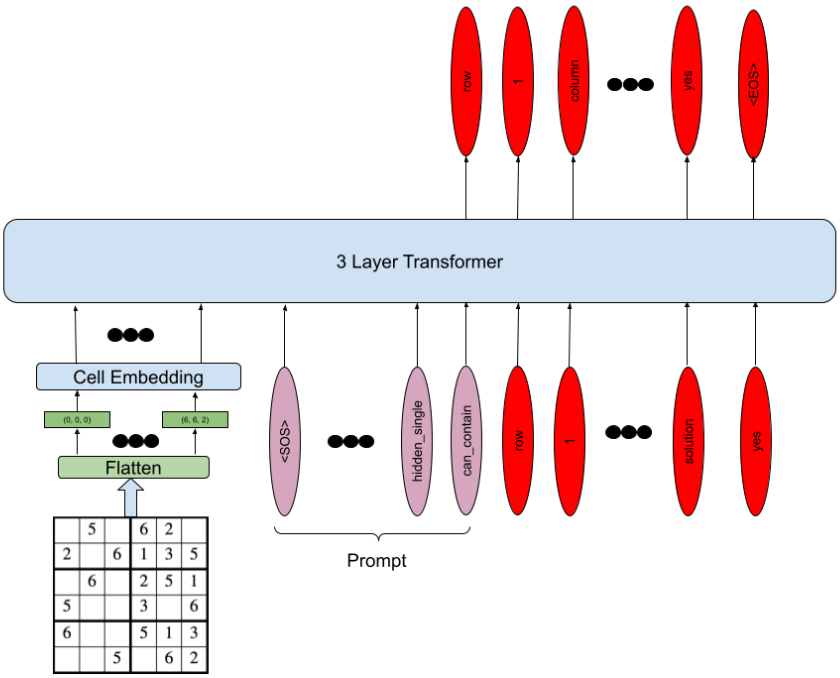


Figure 1: Diagram of the model architecture and how it processes inputs.

layers, and the output vectors are decoded using a linear layer to form the final logit values for the predicted output tokens.

During training, we use teacher-forcing to predict the next token at each sequence position and cross-entropy with the target sequence. We mask the loss so that the loss is only applied after the 'digit' token in the hidden single and full house tasks, and after the column number token in the naked single tasks. The loss is computed using cross-entropy and the model is optimized using Adam [12] with a base learning rate of 0.0001. We keep the same batch size of 192 samples for each stage in the curriculum, performing a weight update at the end of every batch.

### 5.5 Curriculum and Performance Assessment Details

See Figure 2 below for an overview of the curriculum training phase. We do 40k weight updates for the Full House task, 30k for the Can Contain task and 50k for the Hidden Single task. We evaluate OODG performance as measured on a test set of 1,000 OOD Hidden Single task examples. We perform this evalation after every 50 weight updates during the first 1000 weight updates, and after every 250 updates after that. Final OOD accuracy for a run is the accuracy over 40 evaluations occurring within the last 10k updates.

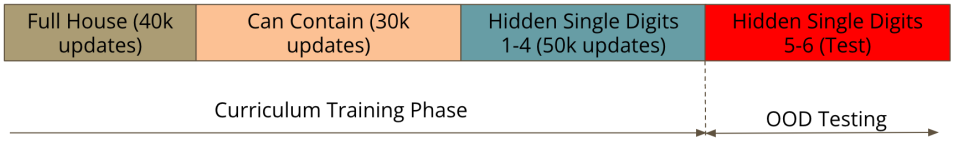


Figure 2: The curriculum used in measuring OODG in the example of holding out two digits from HS puzzles

### 5.6 Shared Digit Representation and Separate Board Architecture Details

To provide a shared digit representation, we re-use the embedding matrix for prompt tokens for computing the digit embedding. Then, we concatenate the row and column embedding for a board cell and add it to the digit embedding to produce the final embedding for a cell.

This was achieved by encoding a board cell as follows:

$$c_e = FF_\theta(D_e(c_{row})|D_e(c_{column})|V_e(c_{digit}))$$

Where $V_e$ is the embedding matrix for text. To have the overall dimension of the cell encoding match the text encoding, we use a feed-forward layer ($FF$) to project the concatenated embedding back into the same output dimension as $V_e$.

To provide a separate board representation, we first compute a key and value embedding for each cell in the board that each decoder layer can then query independently. This is similar to the idea used in the typical transformer decoder [12] - which can perform self-attention over the output of a transformer encoder. A diagram for this architecture is shown in Figure 3.

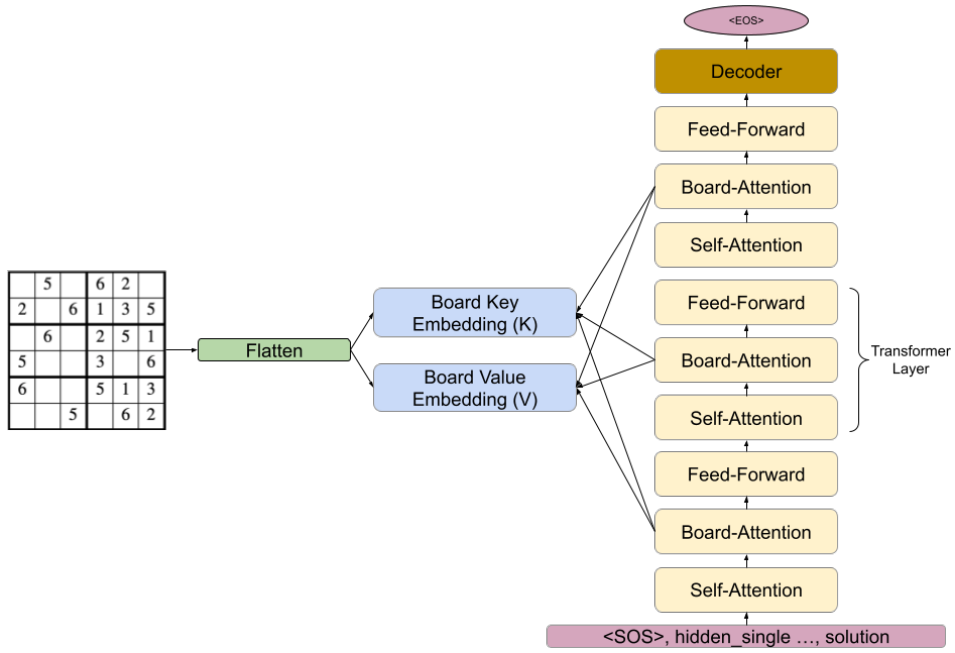


Figure 3: How a shared embedding for the board state is used across all transformer layers

### 5.7 Synaptic Intelligence Details

We use a regularization constant for $c = 0.15$ which determines how strongly the reference weight from past tasks in the curriculum is weighted in the overall loss function.

### 5.8 Out of Distribution Dataset Details

See Table 5 for a full description of the restricted datasets used to test OODG.

### 5.9 Further OODG Results

#### 5.9.1 Four Digit OODG Results

We also ran several experiments on a four digit OOD dataset, results are shown in Table 6.

Table 5: Different OOD tasks for evaluation

| Dataset type | Training Distribution for HS puzzles | Holdout Distribution for HS puzzles |
|---|---|---|
| Two Digit Generalization | Hidden single problems with target digits $\{1, 2, 3, 4\}$ | Test generalization to new digits $\{5, 6\}$ |
| Four Digit Generalization | Hidden single problems with target digits $\{1, 2\}$ | Test generalization to new digits $\{3, 4, 5, 6\}$ |
| Digit and House Type Generalization | Hidden single problems with target digits $\{1, 2\}$ and only *column* or *box* house types | Test generalization on new digits $\{3, 4, 5, 6\}$ and new house type (*row*) |

Table 6: Results for four digit OODG

| Larger TD | Weight Sharing | Repr & Arch | LR Decay | SI | Median Peak OOD Acc | Mean Final OOD Acc | Median Final OOD Acc | Runs over 90% Peak Acc | Mean updates to 90% Peak Acc |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | | | ✓ | | 71.6% | 50.7% | 45.8% | 20% | 6316 |
| ✓ | ✓ | Both | ✓ | | 73.7% | 36.7% | 36.3% | 0% | N/A |
| ✓ | | | | ✓ | 90.0% | 70.2% | 77.2% | 40% | 536 |
| ✓ | ✓ | | | ✓ | 87.9% | 75.5% | 77.9% | 40% | 422 |
| ✓ | | Both | | ✓ | 98.0% | **96.2%** | 95.2% | **100%** | 368 |
| ✓ | ✓ | Both | | ✓ | **98.7%** | 95.4% | **95.8%** | 100% | **206** |

### 5.9.2 Digit and House Type OODG Results

See Figure 4 for a full distribution (10 runs) for each experiment in the digit and house type OODG dataset and Table 4 for accuracy by holdout type.

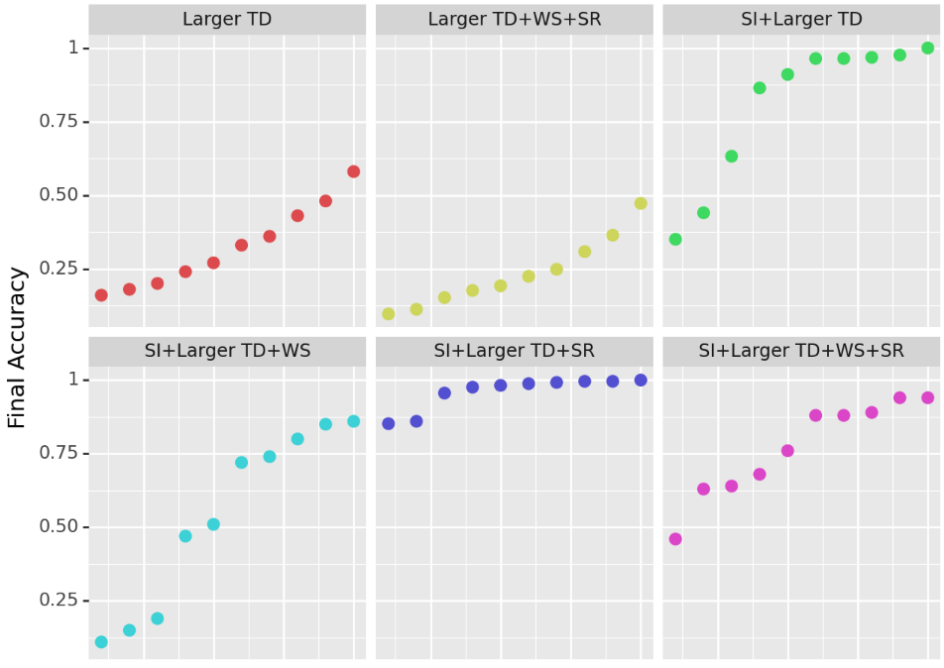


Figure 4: Distribution of actual training runs (10 for each experiment) for the digit and house type holdout dataset

Table 7: Accuracy per holdout dimension for four-digit and house type OODG experiment

| Larger TD | Weight Sharing | Rep & Arch | LR Decay | SI | Median Digits Only Acc | Median House Type Only Acc | Median Digits and House Type Acc |
|---|---|---|---|---|---|---|---|
| ✓ | | | ✓ | | 45.8% | 2.4% | 1.6% |
| ✓ | ✓ | Both | ✓ | | 35.4% | 9.5% | 3.0% |
| ✓ | | | | ✓ | 92.5% | 98.1% | 90.5% |
| ✓ | ✓ | | | ✓ | 85.3% | 79.2% | 64.5% |
| ✓ | | Both | | ✓ | **99.0%** | **99.2%** | **97.4%** |
| ✓ | ✓ | Both | | ✓ | 89.3% | 87.7% | 74.9% |