A Appendix

A.1 Architectures

The architectures we use in our experiments are very similar to those described by Schwarzschild et al. [22], but we provide the details here for convenience.

In all the models, the first layer p is a convolutional layer with 3×3 filters (or filters of length three in the 1D case) followed by a ReLU non-linearity, which projects the input into feature space. Each filter strides by one, and inputs are padded with one unit in each direction. The number of filters is specified per dataset in Table 1 as the "width" of the network. The internal blocks are standard residual blocks with four convolutional layers (followed by ReLUs) and skip connections every two layers [10]. These blocks share weights in recurrent networks and are simply repeated with distinct parameters in feed-forward models. Models with recall have additional convolutional layers that map the concatenated inputs and features $[\phi, x]$ to the shape of the feature maps ϕ. The feature maps have the same spacial dimension as the input and w channels, where w denotes the width of the model. The final block h is composed of three convolutional layers with decreasing widths (specific numbers are in the Table 1), and ReLUs after the first two. The third and final convolutional layer in h has two channel outputs used for binary pixel classification.

Table 1: Model architectures for all main experiments. Note, we perform ablations where we change the width or the maximum number of iterations and those parameters are indicated where appropriate.

Dataset	Width	# Channels in h layers
Prefix Sums	400	400, 200, 2
Mazes	128	32, 8, 2
Chess	512	32, 8, 2

A.2 Computational resources

We use Nvidia GeForce RTX 2080Ti GPUs for all of our experiments. The models that solve prefix sums train in under an hour on a single GPU. The maze models take up to eight hours to train on one GPU. Finally, the chess puzzle networks were trained using four GPUs at once and can take between 24 and 48 hours.

A.3 The loss

Prefix sum problems with n bits are input to the model as a vector $x \in \{0,1\}^n$, and the output is denoted by $\hat{y} \in \mathbb{R}^{n\times 2}$. The target output is $y \in \{0,1\}^n$. We compute the loss as follows.

$$\ell(\hat{y}, y) = -\frac{1}{n}\sum_{i=0}^{n-1} \log \frac{e^{\hat{y}[i,y_i]}}{e^{\hat{y}[i,0]} + e^{\hat{y}[i,1]}} \tag{3}$$

where $[\cdot,\cdot]$ indexes the output array. Therefore, for a batch of B instances, the total loss is

$$\mathcal{L}(\hat{y}, y) = \frac{1}{B}\sum_{b=0}^{B-1} \ell(\hat{y}[b], y[b]). \tag{4}$$

This loss applies to all three problem types we consider. Note that a maze represented by an $n \times n \times 3$ input has n^2 pixels and the loss can be averaged over those pixels. The same applies to chess, where there are always 64 pixels.

When computing the progressive loss in Algorithm 1, we compute $\mathcal{L}_{\text{max\_iters}}$ and $\mathcal{L}_{\text{progressive}}$ as follows.

$$\mathcal{L}_{\text{max\_iters}} = \mathcal{L}(\hat{y}_m, y) \text{ and } \mathcal{L}_{\text{progressive}} = \mathcal{L}(\hat{y}_k, y) \tag{5}$$

Note, in Algorithm 1, when the weight α is equal to 1, there is no contribution from the full forward pass and also that $\alpha = 0$ corresponds to full backpropagation through all iterations with no incremental objective.

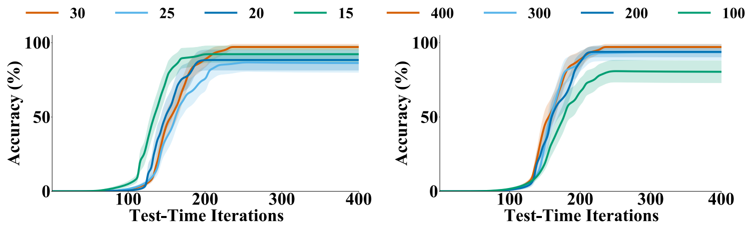

Figure 8: **Top:** Accuracy on 512-bit strings for different values of m show that $m = 30$ is sufficient. **Bottom** Accuracy on 512-bit strings for different values of w. The gain plateaus on prefix sum models at $w = 400$. Models presented here are trained with $\alpha = 1$ and recall.

A reviewer has brought up the interesting comparison of our method to a loss that directly penalizes the run time by weighting the loss with increasing coefficients as a function of iteration. We have run some preliminary experiments, where we find models trained this way exhibit worse generalization to test sets which are harder than their training data. Specifically, on prefix sums, these models solve only $72.45 \pm 8.91\%$ of 512-bit inputs. On maze data, these models were not able to solve any 59x59 mazes in our test set, and we note that the training accuracy is >99% and the accuracy on 33x33 mazes is 93.8%, indicating that training was successful. We believe a more thorough exploration of Deep Thinking networks with variants of such run-time penalties would be interesting future work.

A.4 Hyperparameters

We describe the training set-up for each experiment and discuss hyperparameter choices.

All prefix sum models are optimized using Adam [14] and gradient clipping. Maze models are also trained with Adam, but no clipping is used. Chess models train stably with SGD without gradient clipping. All training is done with a weight decay coefficient of 0.0002 and training with SGD uses a momentum coefficient of 0.9.

In each training run, we hold out 20% of the training data to use as a validation set, and we save for testing the checkpoint with the highest validation accuracy.

Coarse experimentation with learning rates and decay schedules is used to determine optimal choices of these hyperparameters for reproducibility and speed of training. Models were trained with exponential warm-up for a small number of epochs before the main training routine.

In Table 2, we present the training hyperparameters shared among models presented in Section 4 for each problem instance.

Table 2: Training hyperparameters. Dashes indicate that we did not utiltize those options.

Task	Optim.	Learning Rate	Decay Schedule	Decay Factor	Warm-Up	Epochs	Clip
Prefix Sums	Adam	0.001	[60, 100]	0.01	10	150	1.0
Mazes	Adam	0.001	-	-	10	50	-
Mazes (FF)	Adam	0.0001	[175]	0.1	10	200	-
Chess Puzzles	SGD	0.010	[100, 110]	0.01	3	120	-

We perform width and depth experiments to demonstrate that our best prefix sum models are roughly on a plateau in hyperparameter space with respect to performance on 512-bit data. We see from Figure A.4 that performance increases steadily with depth, but no additional performance is achieved by increasing m beyond 30. From Figure 8, we also see performance steadily increases with network width. We choose to use a width of 400 filters per layer and $m = 30$ because this adequately solves 512-bit data.

These experiments are performed on prefix sums (since they are realtively cheap to run) to exemplify the general correlation between depth, width, and performance observed on all tasks. The choice of depth and width of maze and chess models is also made to be only as large as necessary to avoid excessive memory and computational costs.

We determine the optimal weight to be used it the combination of the loss terms using a coarse grid search. For prefix sums we test models with α values in $[0, 0.25, 0.5, 0.75, 1]$, for mazes we test values $[0, 0.01, 0.1, 0.5, 1]$, and for chess we test values $[0, 0.5, 1]$.

A reviewer has pointed out the possibility of exploring a modified version of progressive loss, where n in Algorithm 1 is always equal to zero. Without any further modification, this would mean k is sampled from the larger interval $\{1, m\}$ and would require more computation on average due to the increased expectation for number of backward passes. However, one could then sample k from a smaller interval to match the expectation in the case where n is sampled from $\{1, m\}$, which would save computation on forward passes. We believe such optimizations could be an interesting avenue for future work, but we note that for the case of prefix sums a preliminary evaluation of the expensive version of the $n = 0$ method produces a less effective learned algorithm. Specifically, DT nets with recall achieve $90.27 \pm 5.95\%$ on 512-bit data with the $n = 0$ loss, and with the sampling of n used in Algorithm 1, they achieve $97.12 \pm 1.88\%$.

A.5 Extended results

Tables 3, 4, and 5 show the peak accuracy of each of the curves presented in the plots in Section 4.

Since we aim to learn an algorithm for each of the three tasks (*i.e.* the model should work for all training examples), we use a strict *training* accuracy convergence criteria. Models that do not reach a training accuracy of at least 99% are considered unconverged and are removed from the averages in the tables below to maintain fair comparisons across the various experimental groups. For prefix sums, 6 out of 20 of the models in the progressive loss with no recall category were filtered out this way, and no models were filtered for the other categories. For mazes, 1 out every 5 models trained with recall were filtered out this way, and no models were filtered the other three categories. For all categories, about half of the chess models did not converge. We would like to stress that this filtering was done *solely* based on training accuracy, which is standard practice.

We would like to highlight that the additional performance gained from the recall architecture is *not* due to the marginally higher parameter count. Schwarzschild et al. [23] explore the effect of additional depth on the performance of vanilla DT nets, and the increased performance of DT nets with recall *far* outweighs the benefit from additional capacity of deeper or wider models.

Table 3: The peak accuracy and corresponding test-time iteration number for prefix sum solving model performance curves in Figures 6 and 3.

Tested on 48-bit Strings			
Model	α	Peak Iter.	Peak Acc. (%)
DT	0.0	42	94.61 ± 1.19
DT	1.0	27	97.73 ± 1.80
DT-Recall	0.0	46	99.97 ± 0.01
DT-Recall	1.0	26	99.96 ± 0.02
FF	0.0	30	27.15 ± 2.56

Tested on 512-bit Strings			
Model	α	Peak Iter.	Peak Acc. (%)
DT	0.0	-	0.00 ± 0.00
DT	1.0	171	11.26 ± 6.90
DT-Recall	0.0	466	96.19 ± 3.73
DT-Recall	1.0	237	97.12 ± 1.88
FF	0.0	30	0.00 ± 0.00

Table 4: The peak accuracy and corresponding test-time iteration number for maze solving model performance curves in Figures 6 and 4.

Tested on 13 × 13 Mazes			
Model	α	Peak Iter.	Peak Acc. (%)
DT	0.00	40	85.59 ± 2.81
DT	0.01	38	86.08 ± 3.96
DT-Recall	0.00	94	99.94 ± 0.03
DT-Recall	0.01	70	99.88 ± 0.05
FF	0.00	0	38.22 ± 5.28

Tested on 59 × 59 Mazes			
Model	α	Peak Iter.	Peak Acc. (%)
DT	0.00	-	0.00 ± 0.00
DT	0.01	-	0.00 ± 0.00
DT-Recall	0.00	999	82.72 ± 15.14
DT-Recall	0.01	984	97.30 ± 0.68
FF	0.00	-	0.00 ± 0.00

Figure 9 shows the test performance of chess models on even harder test sets. It is interesting to note that recall and our loss both still help dramatically, but there is an apparent ceiling on

Table 5: Peak accuracy and iteration number for chess puzzle performance curves in Figure 5.

Tested on puzzles 600K-700K			
Model	α	Peak Iter.	Peak Acc. (%)
DT	0.0	32	78.81 ± 1.04
DT	0.5	43	82.32 ± 0.10
DT-Recall	0.0	29	82.12 ± 0.69
DT-Recall	0.5	57	82.69 ± 0.27
FF	0.0	30	61.17 ± 1.76

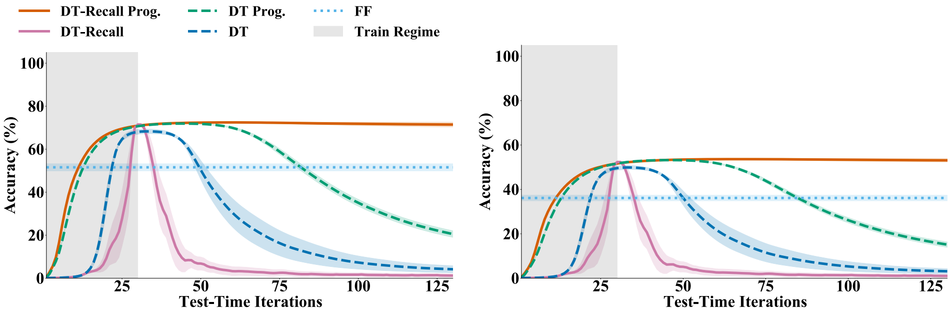

Figure 9: Chess performance when tested on puzzles with indices from 700k to 800k (left) and from 1M to 1.1M (right).

generalizing from the easy puzzles to much harder ones. We leave investigation into this phenomenon for future work.

A.6 In-distribution results

Information on how models perform in distribution reveals that each class of model fits the training data and generalizes in the classical sense. Specifically, we report the accuracy on a held-out validation set from the same problem difficulty as the training data. For recurrent models, we evaluate the performance using the maximum number of iterations used during training, irrespective of the training objective. See Table 6.

A.7 Hard to easy

One natural query about our models is how well they perform on test sets that are easier/smaller than the data used for training. Figure 10 shows that recurrent models can generally solve easier/smaller problems in fewer iterations. Even though they solve the easier problems in fewer than the maximum number of iterations used in training, we still see that models without recall suffer from overthinking.

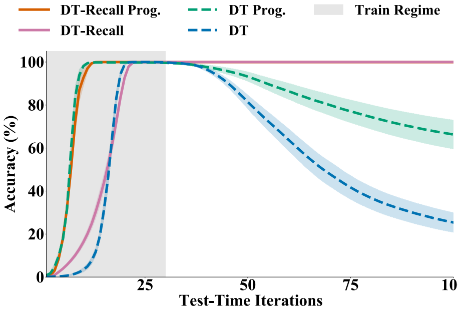

Figure 10: Accuracy as a function of iteration for prefix sum models when generalizing from harder 32-bit strings to easier 24-bit strings.

Table 6: Training accuracy and in-distribution validation accuracy for models presented in Figures 3 and 6.

Trained on 32-bit Prefix Sum Problems			
Model	α	Train Acc. (%)	Valid Acc. (%)
DT	0.00	99.98 ± 0.01	100.00 ± 0.00
DT	1.00	99.57 ± 0.38	97.73 ± 1.80
DT-Recall	0.00	99.95 ± 0.02	100.00 ± 0.00
DT-Recall	1.00	99.98 ± 0.01	100.00 ± 0.00
FF	0.00	99.76 ± 0.14	99.76 ± 0.14

Trained on 9×9 Mazes			
Model	α	Train Acc. (%)	Valid Acc. (%)
DT	0.00	100.00 ± 0.00	100.00 ± 0.00
DT	0.01	99.98 ± 0.01	99.98 ± 0.01
DT-Recall	0.00	99.91 ± 0.07	99.92 ± 0.06
DT-Recall	0.01	99.77 ± 0.15	99.74 ± 0.17
FF	0.00	99.94 ± 0.01	99.93 ± 0.05

Trained on Chess Puzzles 0-600K			
Model	α	Train Acc. (%)	Valid Acc. (%)
DT	0.00	99.98 ± 0.02	92.94 ± 0.34
DT	0.50	99.90 ± 0.01	94.16 ± 0.02
DT-Recall	0.00	99.77 ± 0.02	94.16 ± 0.03
DT-Recall	0.50	99.34 ± 0.02	94.18 ± 0.07
FF	0.00	96.89 ± 0.58	83.24 ± 1.22

A.8 More on the overthinking problem

We present additional plots and tables to give a more complete picture of how some models overcome the overthinking problem presented in Section 5. Table 7 shows the results from several experiments. The first column indicates the average number of iterations to solve a maze and the best accuracy (over all the test iterations considered). Then, we examine the behavior when the features are perturbed after the first iteration by adding mean zero and variance one Gaussian noise ('Noise' column) and we find that this completely destroys the models without recall while only slightly slowing down those with recall. Next, we see the same trend when we replace the features at the first iteration with arrays of zeros ('Zeros' column). Finally, we swap the features after 50 iterations with those obtained from solving an entirely different maze, and again the models without recall cannot recover from this, but those with recall achieve near perfect accuracy. Figures 11-13 shows accuracy curves for these experiments. Figure 14 shows the norm of the difference in features when we input random noise instead of actual mazes. This plot shows that the stable behavior of our models is not limited to the portion of input space occupied by real mazes, rather the process they learn exhibits convergent behavior even on random inputs.

Similar experiments on prefix sum computation reveal that recurrent models with recall seem to converge as well.

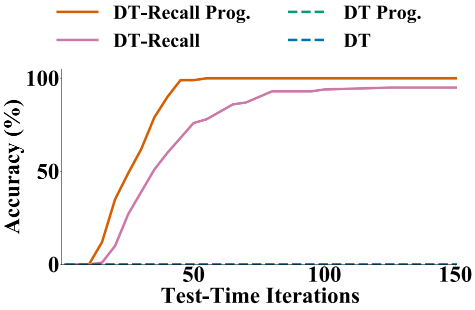

Figure 11: Maze solving model performance when Gaussian noise is added to the features after the first iteration. DT models with recall are robust to this perturbation.

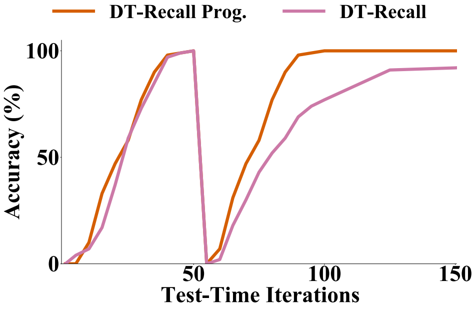

Figure 12: Maze solving model performance when the features after 50 iterations are replaced with zeros. DT models with recall are also robust to this perturbation.

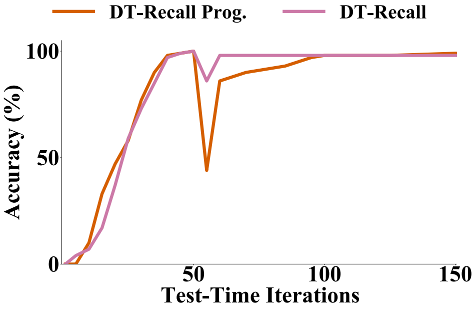

Figure 13: Maze solving model performance when the start and end of the maze are changed to be closer together after 50 iterations. DT models with recall can self-correct.

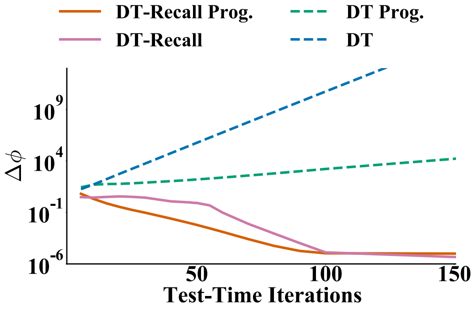

Figure 14: The change in the norm of the feature maps for maze solvers as function of iteration shows that models with recall seem to move the feature maps less and less with each iteration.

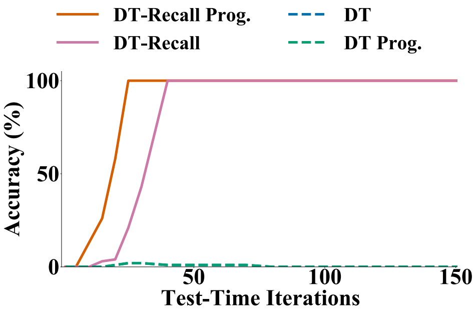

Figure 15: Prefix sum model performance when Gaussian noise is added to the features after the first iteration. DT models with recall are robust to this perturbation.

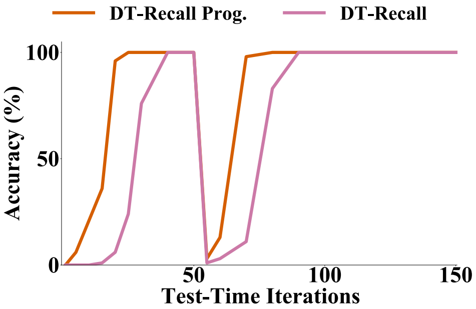

Figure 16: Prefix sum model performance when the features after the first iteration are replaced with zeros. DT models with recall are also robust to this perturbation.

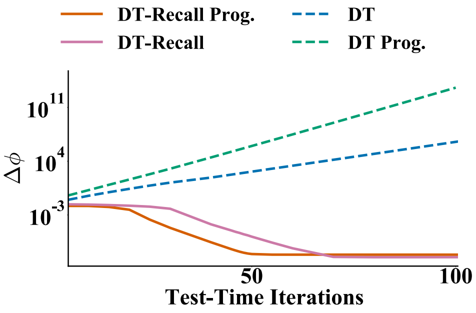

Figure 17: The changes in the norm of the feature maps for prefix sums as function of iteration show that models with recall seem to move the feature maps less and less with each iteration.

Table 7: Average iterations to solution, with the peak accuracy in parentheses. We evaluate maze solving models on 13×13 mazes, where we intervene in the solving process in different ways.

Model	Clean	Noise	Zeros	Swapped
DT	27.68 (87 %)	- (0%)	- (0%)	- (0%)
DT w/ New Loss	16.87 (95 %)	- (0%)	- (0%)	- (0%)
DT-Recall	23.54 (100 %)	35.70(100 %)	35.04 (96 %)	33.92 (97 %)
DT-Recall w/ New Loss	22.10 (100 %)	25.10(100 %)	22.48 (100 %)	28.79 (100 %)

A.9 A fun maze

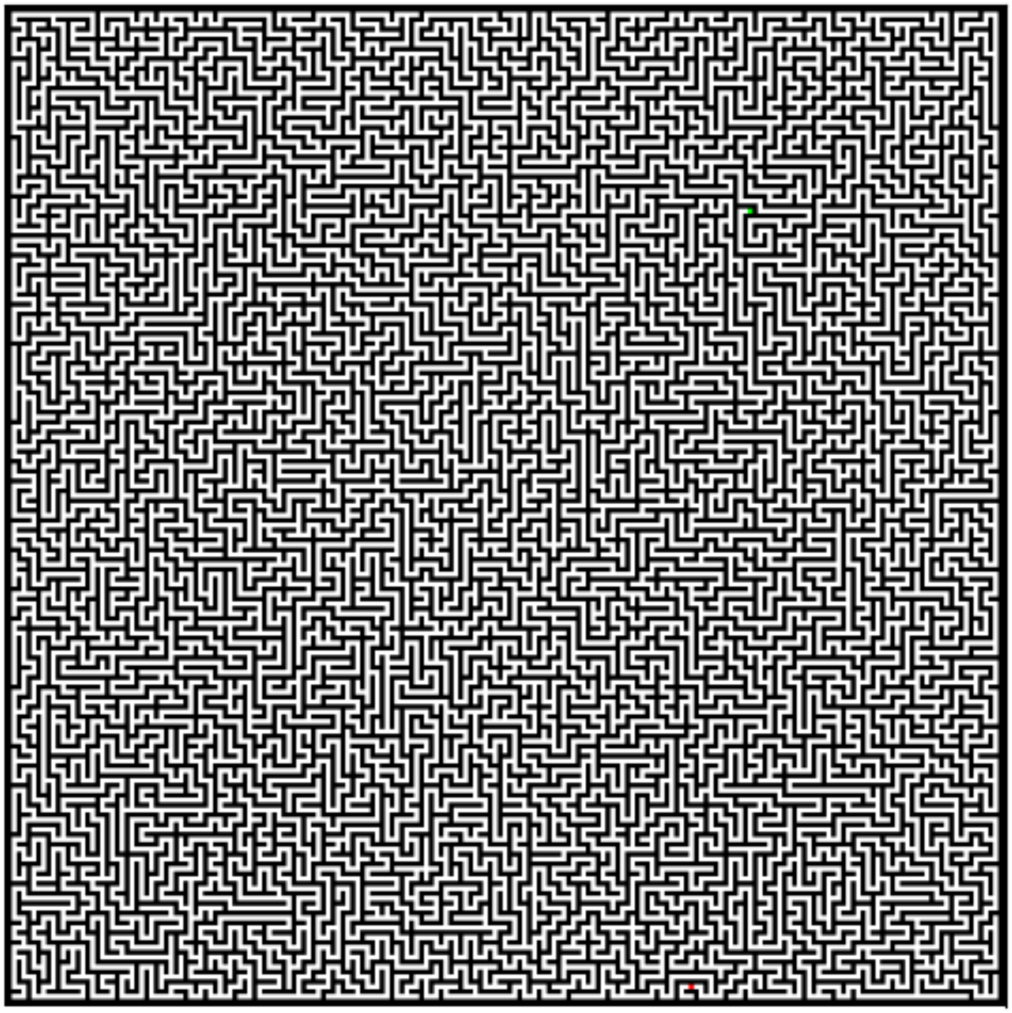

Figure 18: Fun for the whole family! We leave solving this 201×201 maze as an exercise for the reader. The green starting dot is near the center of the upper right quadrant. The red ending point is on the second row from the bottom.

A.10 A ridiculous maze

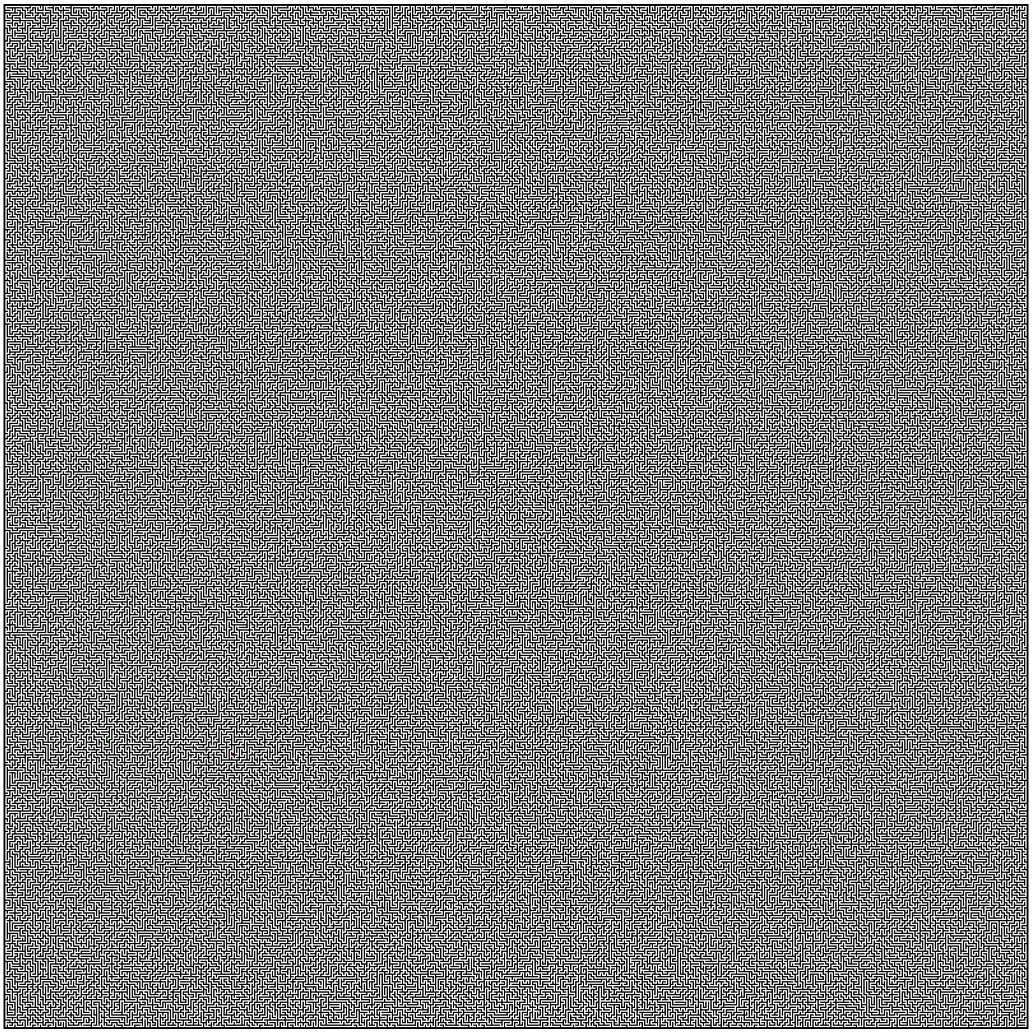

Figure 19: An example of a *very, very* fun 801×801 maze. The green starting dot is three quarters of the way down on the left-hand side. A model trained using only 9×9 mazes and 30 iterations is able to solve this extreme maze at test time using 10,000 iterations, which is equivalent to 100,004 convolutional layers.