A. Hyperparameter Settings

Unless otherwise noted, all models in Tables 2, 1, 3, 4, 7, 6, 5, 8 are trained on 2D mazes of size 15×15 for 30 epochs using a learning rate of 1e-3, batch size 32, gradient clipping of 40, and 25k/5k/5k train-val-test split. An initial sweep of learning rates over { 5e-3, 1e-3, 1e-2, 1e-1 } and found that 1e-3 worked well for both VIN and GPPN. We do a hyperparameter sweep of (K, F) over $K \in \{5, 10, 15, 20, 30\}$ and $F \in \{3, 5, 7, 9, 11\}$. The only exceptions are the following: For the larger 28×28 maze in Table 7, we sweep K over $\{14, 28, 56\}$ to account for longer trajectories required to solve some mazes. For the 10k and 100k dataset sizes in Table 4, we used a train-test-val split of 10k/2k/2k and 100k/10k/10k, respectively. The variance results in Table 5 are obtained using dataset size 100k. The 3D ViZDoom results in Table 8 were obtained using $K = 30$, the best setting of F for each transition kernel, a smaller dataset size 10k , a smaller learning rate 5e-4, and 100 training epochs.

The particular form of the LSTM update the GPPNs take in the experimental section is slightly different from the standard one. The first difference is that we remove the dependence of each layer of the ConvLSTM on the "reward" function $\bar{R}_{[i',j',F]}$ since we did not find this skip-connection helped performance much in preliminary experiments. Therefore layers of the GPPN only take as input the previous layer's hidden units. The second change was made to make the GPPN easier to implement in a framework where built-in LSTM updates are available but ConvLSTMs are not. It first takes the convolution over the previous hidden layers and produces a 1-channel feature map, and then for each position passes that feature map position to the framework's built-in LSTM update. This is similar to having a single shared input gate for all the inputs. When tested against a GPPN with the standard LSTM update equation with full input gating, we did not observe any significant different in test metrics but the single input gate did save some computation time.

B. Learning Plots

As mentioned in Section 5.5, we provide additional learning plots for varying dataset sizes (Figure 5), varying maze sizes (Figure 6), and 3D ViZDoom results (Figure 7).

C. Hyper-VIN

The VIN uses convolutions to represent the model, which causes it to effectively be spatially invariant, meaning VINs are incapable of truly solving mazeworld in the same way as value iteration on the true model. The result is that VINs learn a workaround that enables it to deal with non-linearities over the state space: it assigns a large negative reward to every wall position. This is shown in Figure 4: the large reward gradient between walls and non-walls discourages the model from producing policies that "visit" wall states which would be impossible under the true model. Additionally, the spatial convolution model is fixed and invariant for all mazes, which is suboptimal as each MDP in the 2D environments require a different transition kernel based on the maze design.

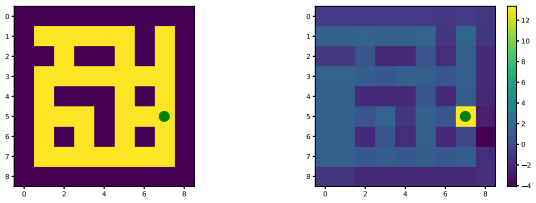

Figure 4. **Left**: A sample 2D maze environment, where yellow cells, purple cells, and the green circle represent open spaces, walls, and the goal state respectively. All mazes are constructed as fully connected trees with a decimation parameter that destroys walls with a certain probability. **Right**: The initial reward vector learned for the 2D maze task on a fully trained VIN. VIN gets around the spatial invariance of its model by applying a large negative reward to states that should never be entered (walls) and a large reward for the goal location.

In this section, we try to alleviate this issue by, first, untying the weights of the spatial convolution and, second, predicting the untied convolution weights directly from the maze design. We call this variant the Hyper-VIN, adopting the naming convention from HyperNetworks (Ha et al., 2017) which also used the kernel of using a network with weights predicted from another network. To implement the Hyper-VIN, we predict for each position (i, j) in the environment a convolutional weight matrix from the input map design. The Hyper-VIN update equation then becomes:

$$\bar{V}^{(t)}_{i',j'} = \omega \left(W_R^{\bar{a},i',j'} \bar{R}_{[i',j',3]} + W_V^{\bar{a},i',j'} \bar{V}^{(t-1)}_{[i',j',3]} \right)$$

A question that can be asked about Hyper-VIN is if they perform as well as (or better than) the actual algorithms they were designed to mimic because the true algorithm is within the model class. This would provide some evidence whether such modules were actually computing the value, or whether they acted simply like recurrent networks and computed a less interpretable internal representation. Empirically, we instead found that Hyper-VINs have high variance in training and were often difficult to optimize (see Table 9). We can see that Hyper-VINs trained by SGD often fail to reach the performance of their exact algorithmic counterpart (value iteration) on small mazes even though value iteration is within the hypothesis class of these models, suggesting that the optimization of such architectures is significantly difficult.

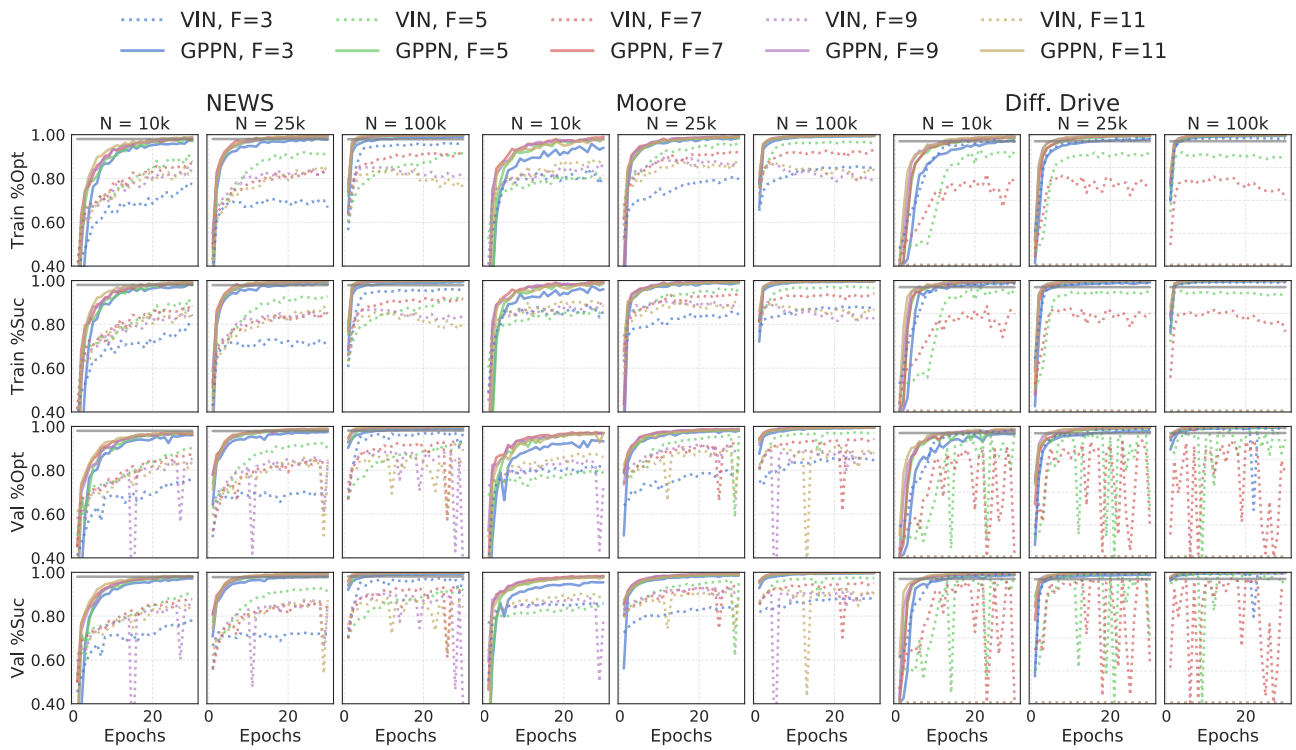

Figure 5. Performance on 2D mazes of size 15×15 with **varying dataset sizes** N. All models are trained using $K = 30$ and learning rate 1e-3.

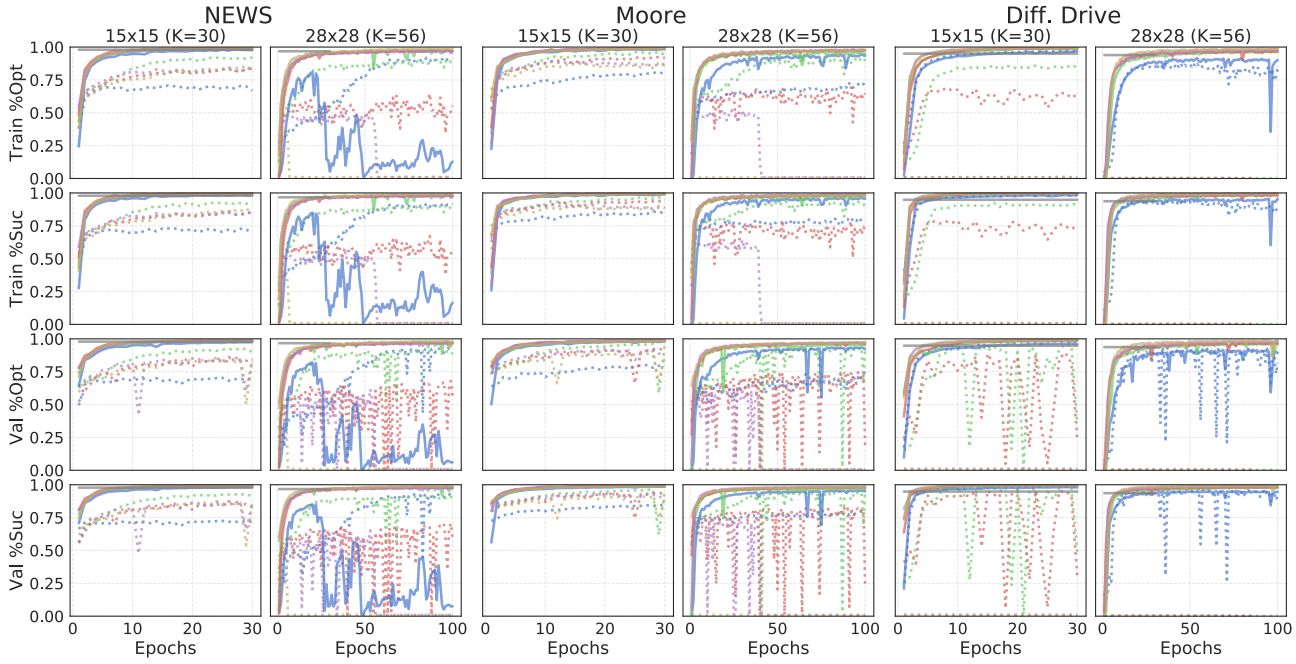

Figure 6. Performance on 2D mazes with **varying maze sizes** $m \times m$. All models are trained using learning rate 1e-3, dataset size 25k, and $K = 30$ (for $m = 15$) or K=56 (for $m = 28$).

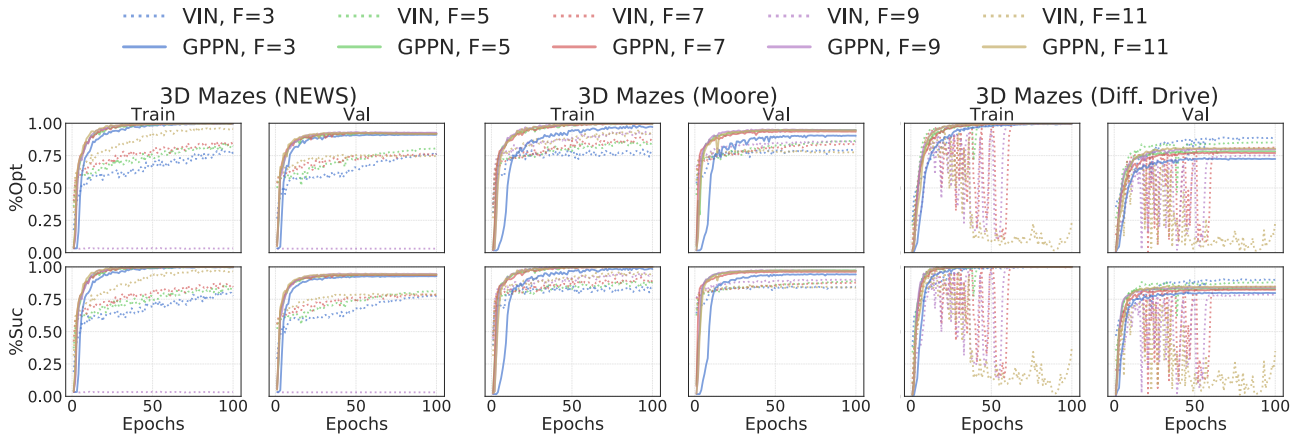

Figure 7. Performance on 3D ViZDoom mazes of size 15×15. All models are trained using K=30, learning rate 5e-4, and dataset size 10k.

Table 9. Validation set performance (mean and standard deviation) on 2D mazes of size 15×15, taken over 7 runs on the same dataset. These results were attained using iteration count $K = 20$ for all models, filter size $F = 3$ for VIN and Hyper-VIN, and $F = 11$ for GPPN. Due to GPU memory limitations with Hyper-VIN, all models were trained using half the hidden dimension compared to experiments in the main paper. Hyper-VIN has high variance in training and is difficult to optimize.

Model	NEWS				Differential Drive			
	%Opt		%Suc		%Opt		%Suc	
	mean	stdev	mean	stdev	mean	stdev	mean	stdev
Hyper-VIN	75.1	11.4	80.2	9.5	77.6	1.9	94.8	0.6
VIN	85.8	6.6	87.1	5.7	97.8	0.1	98.7	0.1
GPPN	98.9	0.2	99.3	0.2	98.1	0.9	98.8	1.1
VI	94.2	-	94.2	-	85.1	-	85.1	-