

---

# Skill-Driven Neurosymbolic State Abstractions: Supplementary Material

---

Anonymous Author(s)

Affiliation

Address

email

## 1 H Network Details

```
2 MSAFlat(  
3     (encoder): Sequential(  
4         (0): Linear(in_features=1568, out_features=64, bias=True)  
5         (1): ReLU()  
6         (2): Linear(in_features=64, out_features=64, bias=True)  
7         (3): ReLU()  
8         (4): Linear(in_features=64, out_features=2, bias=True)  
9     )  
10    (inverse_fc): Sequential(  
11        (0): Linear(in_features=4, out_features=64, bias=True)  
12        (1): ReLU()  
13        (2): Linear(in_features=64, out_features=64, bias=True)  
14        (3): ReLU()  
15        (4): Linear(in_features=64, out_features=5, bias=True)  
16    )  
17    (density_fc): Sequential(  
18        (0): Linear(in_features=4, out_features=64, bias=True)  
19        (1): ReLU()  
20        (2): Linear(in_features=64, out_features=64, bias=True)  
21        (3): ReLU()  
22        (4): Linear(in_features=64, out_features=1, bias=True)  
23    )  
24    (decoder): Sequential(  
25        (0): Linear(in_features=2, out_features=64, bias=True)  
26        (1): ReLU()  
27        (2): Linear(in_features=64, out_features=64, bias=True)  
28        (3): ReLU()  
29        (4): Linear(in_features=64, out_features=1568, bias=True)  
30    )  
31    (mi): Sequential(  
32        (0): Linear(in_features=4, out_features=64, bias=True)  
33        (1): ReLU()  
34        (2): Linear(in_features=64, out_features=64, bias=True)  
35        (3): ReLU()  
36        (4): Linear(in_features=64, out_features=1, bias=True)  
37    )  
38 )
```

We used a two-layered multi-layer perceptron (MLP) with 64 hidden units and ReLU activations for the MSA encoder. For convenience of visualization, we set the output dimensionality of MSA models to two. `inverse_fc` and `density_fc` modules compute the inverse model and the density ratio losses, respectively. `mi` is trained separately (with a separate optimizer) to estimate the mutual information between two hidden units, and the main MSA model backpropagates through this estimate to minimize it. Lastly, we use the decoder only for visualization purposes and do not backpropagate any error through it, which would otherwise make it a reconstruction-based embedding.

```

46 QNetwork(
47     (features_extractor): CombinedExtractor(
48         (extractors): ModuleDict(
49             (achieved_goal): Flatten(start_dim=1, end_dim=-1)
50             (desired_goal): Flatten(start_dim=1, end_dim=-1)
51             (observation): Flatten(start_dim=1, end_dim=-1)
52         )
53     )
54     (q_net): Sequential(
55         (0): Linear(in_features=4704, out_features=64, bias=True)
56         (1): ReLU()
57         (2): Linear(in_features=64, out_features=64, bias=True)
58         (3): ReLU()
59         (4): Linear(in_features=64, out_features=4, bias=True)
60     )
61 )

```

We used the default structure of the DQN model that is provided in `stable-baselines3`<sup>1</sup> [2] with the only difference in the output dimensionality that depends on the number of actions in the domain. Notably, the encoder of MSA and the Q network of the DQN has the same number of layers and hidden units.

## I Hyperparameters

We used the default hyperparameters of the DQN model in `stable-baselines3` with the only change of train frequency from four to one, on which the model was performing better (Table 3). For MSA training, we did not perform a hyperparameter search but instead set them to frequently used values in other neural network trainings (Table 1). This does not pose a serious problem for MSA as the training objective is well-defined. We early-stop the training when both the inverse and the density ratio loss do not improve for five epochs.

The main hyperparameters of Algorithm 1 (Table 2) are minimum transition and reward errors,  $\epsilon_T$ ,  $\epsilon_R$ , and their relative improvements after a refinement,  $\Delta\epsilon_T$ ,  $\Delta\epsilon_R$ . Using the transition error sufficed in our domains.  $\epsilon_T$  and  $\epsilon_R$  define thresholds to consider an abstract state as a candidate for refinement. We used  $\Delta\epsilon_T$  and  $\Delta\epsilon_R$  to check whether the clustering decreased the cumulative transition and reward errors. In that sense,  $\epsilon$  values set the granularity level of the abstract MDP whereas  $\Delta\epsilon$  values filter out refinements that does not improve the score due to poor clustering.

In this paper, we focus on the theoretical construction of the framework, and provide an implementation of this idea. We note that this is one specific instantiation. Many parts—ranging from refinement to independence tests—can be realized with various methods. As such, hyperparameters both in this approach and in the compared baseline is open to further optimization for better performance.

## J Compute Resources

The algorithm presented in the paper first trains an MSA network and builds an abstract MDP on top of the trained embeddings. The training time of the MSA network depends on the dataset size and the available chip configuration. The construction of the abstract MDP consists of iterative clustering and independence test steps. We profiled the current implementation to understand the main bottlenecks.

<sup>1</sup><https://stable-baselines3.readthedocs.io/en/master/>

Table 1: MSA hyperparameters

Parameter	Value
Batch size	32
Validation split	0.1
Optimizer	Adam
Learning rate	0.001
$\beta_1$	0.9
$\beta_2$	0.999
$\epsilon$	$10^{-8}$
Negative sampling rate	10
Inverse loss coeff.	1.0
Density ratio loss coeff.	1.0
Smoothness coeff.	1.0
Mutual information reg. coeff.	1.0
Maximum number of epochs	200
Early stopping patience	5

Table 2: Alg. 1 hyperparameters

Parameter	Value
Minimum transition error $\epsilon_T$	
MNIST chain	0.1
MNIST grid	0.1
Visual Maze	None
Montezuma’s Revenge	0.1
Minimum $\Delta\epsilon_T$ for refinement	
MNIST chain	0.5
MNIST grid	0.5
Visual Maze	0.0
Montezuma’s Revenge	0.5
Minimum reward error $\epsilon_R$	None
Minimum $\Delta\epsilon_R$ for refinement	None
$n$ cluster trials	10
Min. samples for refinement	10

Table 3: DQN hyperparameters

Parameter	Value
Batch size	32
Learning start step	100
Optimizer	Adam
Learning rate	0.0001
$\beta_1$	0.9
$\beta_2$	0.999
$\epsilon$	$10^{-8}$
Soft update coeff. ( $\tau$ )	1.0
Discount factor ( $\gamma$ )	0.99
Train frequency (every $k$ step)	1
Gradient steps (every $k$ rollout)	1
Replay Buffer	HER [1]
Number of sampled goals	4
Target update steps	10000
Exploration fraction	0.1
Exploration initial $\epsilon$	1.0
Exploration final $\epsilon$	0.05
Maximum gradient norm	10.0

Two components that contributed to the overall time complexity the most are (1) `torch.cdist`, that computes the Euclidean distance between each pair of vectors in two tensors to estimate  $\epsilon_T$  using  $k$ -nearest neighbor based independence test, and (2) the assignment of each ground sample to the abstract states after clustering. We believe these processes can be further optimized with sampling approximations and with special data structures like  $k$ -d trees to reduce the time spent on finding nearest neighbors.

We used a GPU cluster to train multiple MSA networks in batch and build the abstract MDP locally on a laptop with Apple M1 Pro chip and 16 GB of unified memory. DQN models are trained locally on this same laptop and another with NVIDIA RTX 4070 GPU and Intel Core Ultra 9 Processor 185H. It is difficult to give accurate numbers on the training times as GPUs on the cluster are shared across multiple users. Trainings on the MNIST domain are completed within minutes whereas trainings on the Visual Maze took around 5 to 10 minutes at  $80 \times 60$  resolution, and 30 to 50 minutes on higher resolutions.

## 101 K MSA Embedding Visualizations

102 We visualized the learned embedding spaces in different domains for varying samples sizes in Figures  
 103 1, 2, 3, 4, and 5.



Figure 1: Learned MSA embedding spaces in the Visual Maze domain for different numbers of samples. 2-dimensional positions are the MSA encodings of input images. Frame colors represent the underlying high-level states.

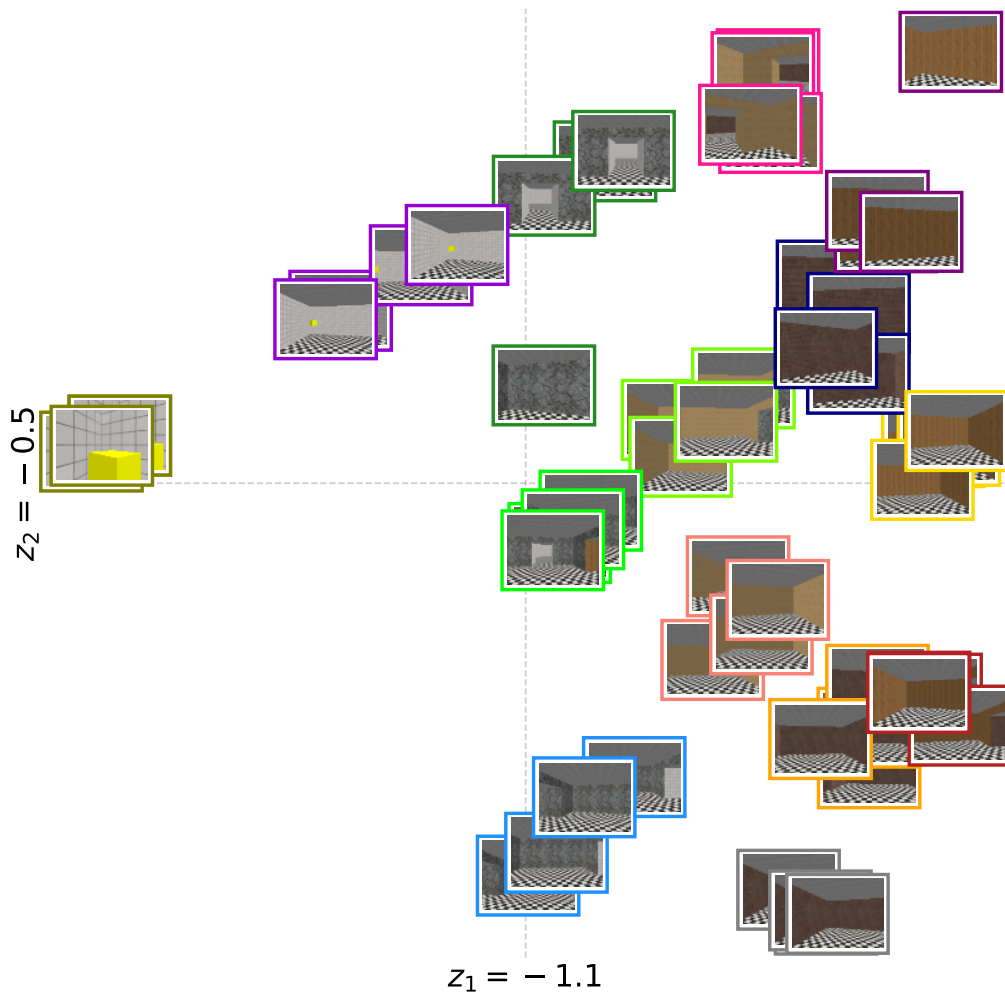


Figure 2: Learned MSA embedding space in the Visual Maze domain at  $160 \times 120$  pixels.

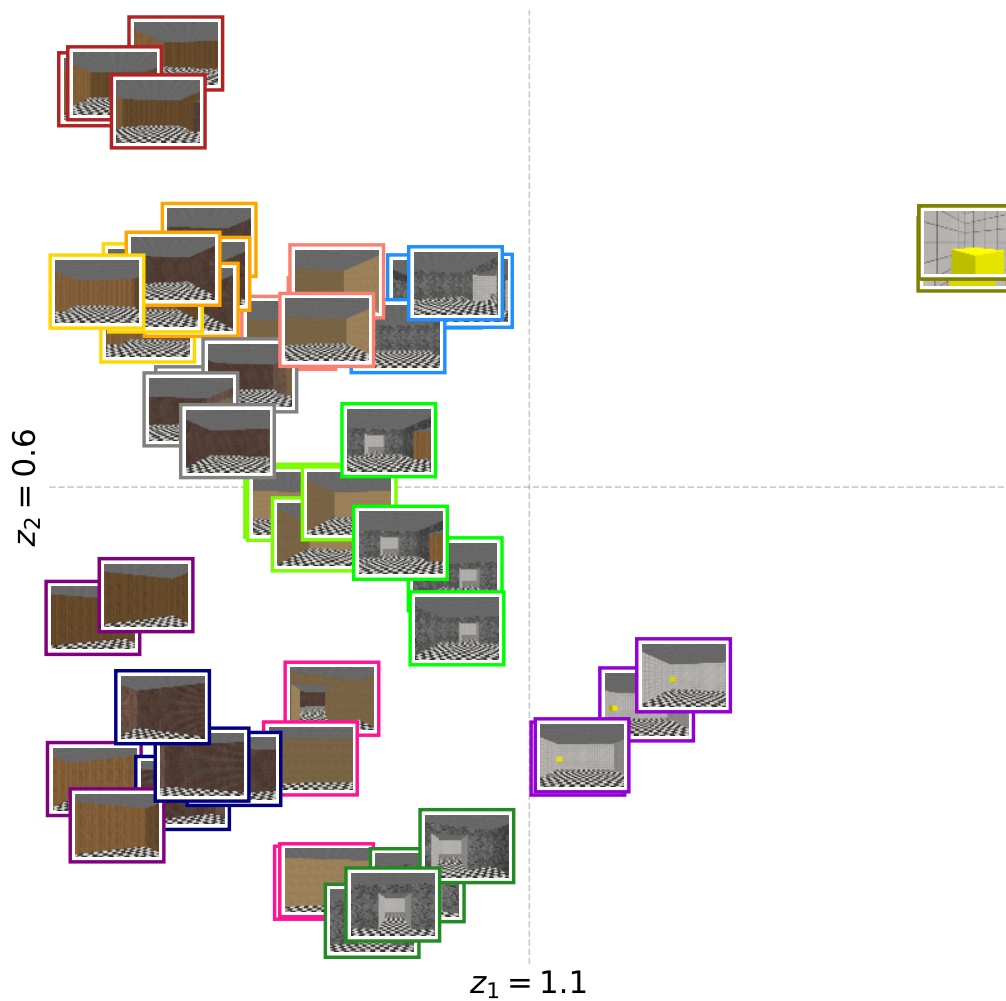


Figure 3: Learned MSA embedding space in the Visual Maze domain at  $240 \times 180$  pixels.

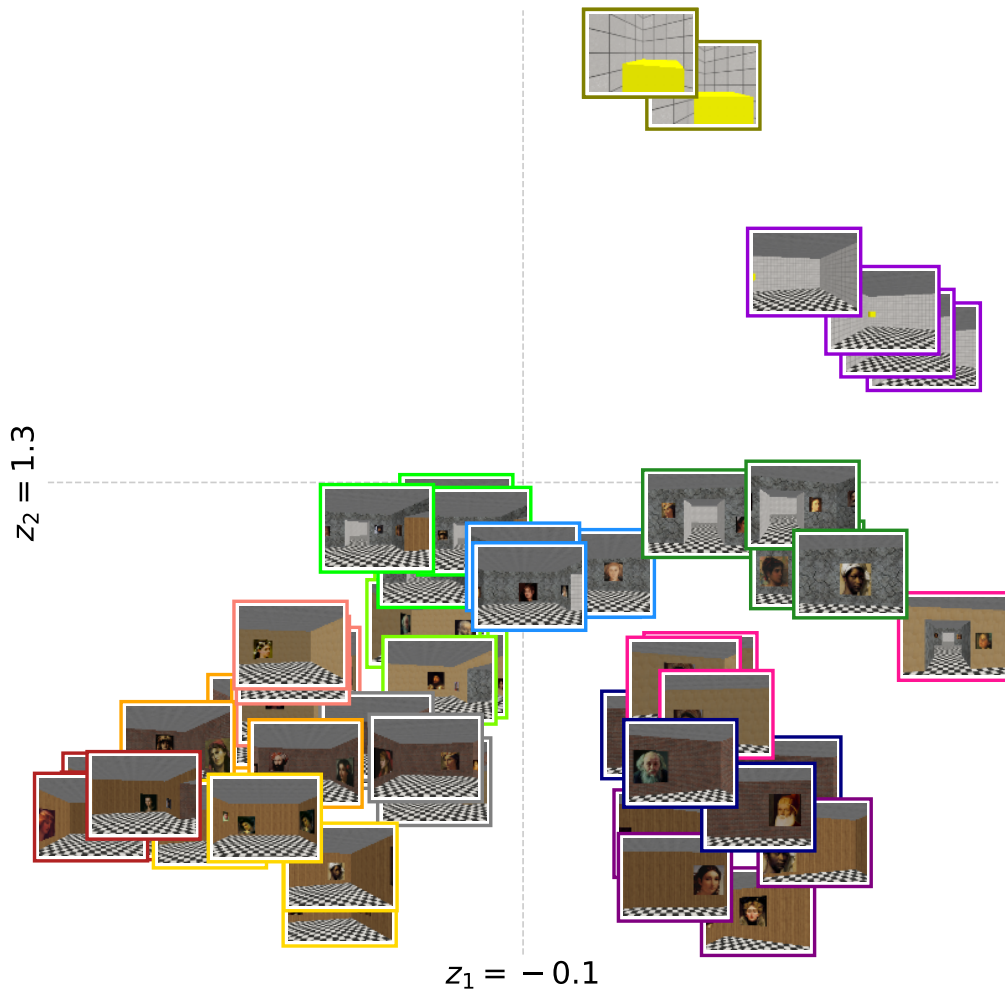
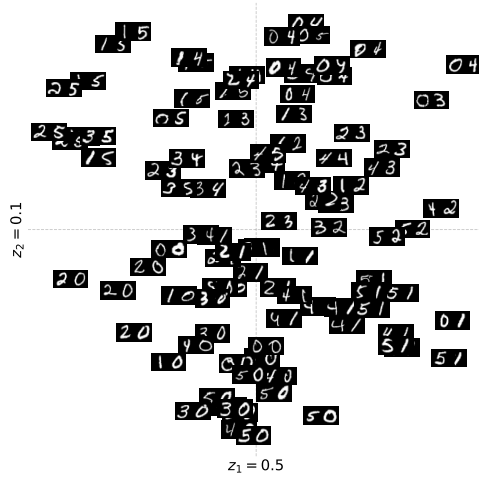
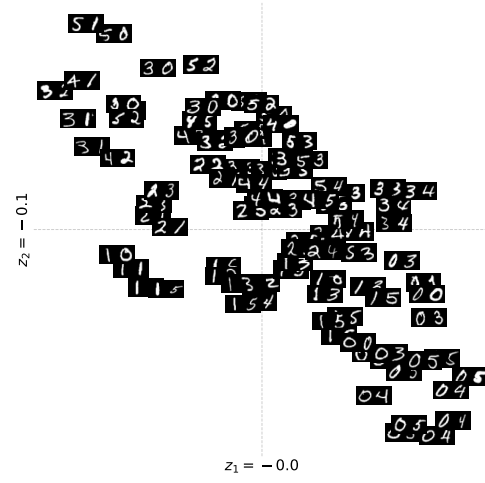


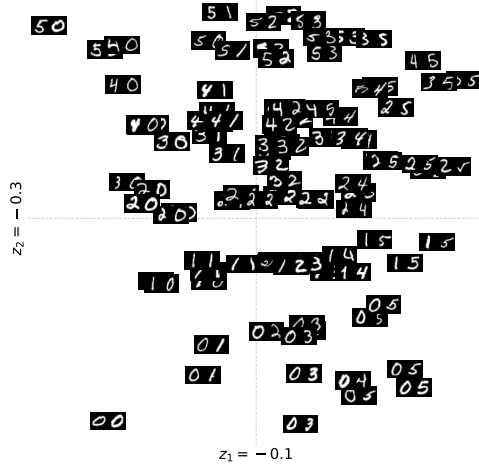
Figure 4: Learned MSA embedding space in the Visual Maze domain at  $240 \times 180$  pixels with random portraits on the wall.



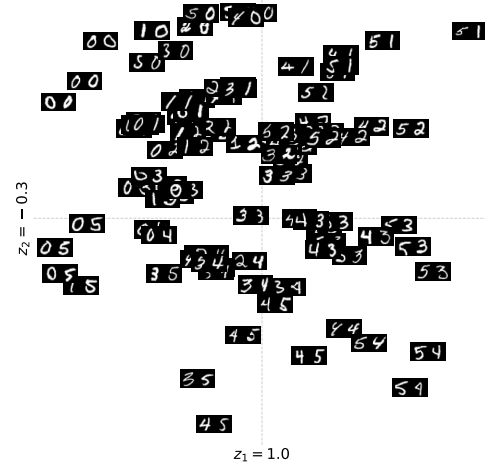
(a) 1000 samples, Seed 0



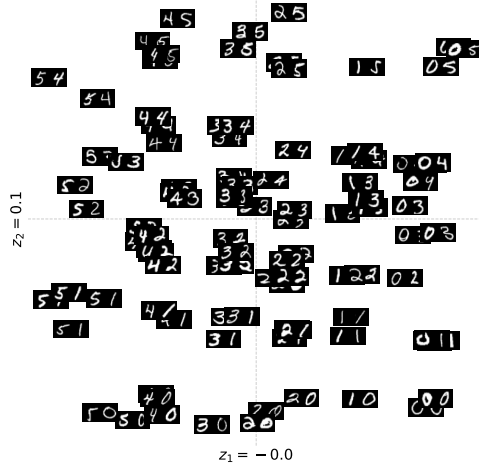
(b) 1000 samples, Seed 1



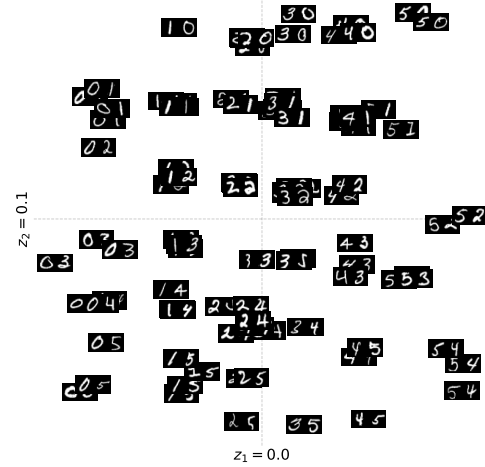
(c) 5000 samples, Seed 0



(d) 5000 samples, Seed 1



(e) 10000 samples, Seed 0



(f) 10000 samples, Seed 1

Figure 5: Learned MSA embedding spaces in the MNIST grid domain for different numbers of samples. 2-dimensional positions are the MSA encodings of input images.



## 104 **References**

- 105 [1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder,  
106 Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay.  
107 In *Advances in Neural Information Processing Systems*, volume 30, 2017.
- 108 [2] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah  
109 Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of*  
110 *Machine Learning Research*, 22(268):1–8, 2021.