Skill-Driven Neurosymbolic State Abstractions

Alper Ahmetoglu[*]
Brown University

Steven James
University of the Witwatersrand

Cameron Allen
UC Berkeley

Sam Lobel
Brown University

David Abel
University of Edinburgh

George Konidaris
Brown University

Abstract

We consider how to construct state abstractions compatible with a given set of abstract actions, to obtain a well-formed abstract Markov decision process (MDP). We show that the Bellman equation suggests that abstract states should represent distributions over states in the ground MDP; we characterize the conditions under which the resulting process is Markov and approximately model-preserving, derive an algorithm for constructing the abstract MDP, and apply it to visual chain and maze tasks. We generalize these results to the factored actions case, characterize the conditions that lead to factored abstract states, and apply the resulting algorithm to a visual grid and Montezuma's Revenge. These results provide a principled, powerful framework for learning neurosymbolic abstract Markov decision processes.

1 Introduction

Reinforcement learning (RL) [62] has achieved remarkable success solving tasks with complex sensorimotor spaces [48, 43]. However, learning and planning at the pixel and motor level, while necessary, must eventually be insufficient: the complexity of real-world tasks such as cooking a meal, inter-city travel, and fixing a car is such that they can only be feasible if decision-making occurs at an abstract level and the low-level details are abstracted away. This is especially true when a single agent must perform *all* these tasks, necessitating a sensorimotor space complex enough to cover all of them. Linking abstract reasoning to low-level action and perception, to enable fast decision-making grounded in real sensorimotor interaction, is therefore critical for general intelligence [23, 38].

Hierarchical RL (HRL) [14] addresses this problem via high-level abstract *actions*, but does not create correspondingly abstract *states*, leaving the agent in its original low-level state space. It would be far preferable to combine *both* types of abstractions to construct an entirely abstract decision process [38], which requires mutually compatible state and action abstractions: those that result in a new, more compact decision process that is Markov and supports near-optimal policies over the abstract actions in the original MDP. Following earlier work in robotics [40], we adopt an *action-first* strategy, constructing state abstractions compatible with a pre-existing set of abstract actions.

We first focus on building an abstract MDP from a ground MDP augmented with a given set of abstract actions, where the Bellman equation suggests that abstract states should represent distributions over ground states. We use this insight to build an abstract decision process, and identify the conditions under which it is Markov and approximately model-preserving [44, 1]. We develop algorithms for constructing the abstraction from data and for planning with it, and apply them to visual chainwalk and maze tasks. We then generalize these results to factored actions, which modify only some state variables in the ground MDP. We characterize the conditions under which this generates factored

[*]Correspondence to Alper Ahmetoglu `aahmetog@cs.brown.edu` and George Konidaris `gdk@brown.edu`.
Extended results are available at `neurosymbolic-mdps.github.io`.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

abstract states, provide an algorithm that constructs the corresponding abstraction, and apply it to a visual gridworld and Montezuma's Revenge, a long-horizon Atari task [15]. These results provide a powerful and principled framework for learning neurosymbolic abstract decision processes.

2 Background

RL problems are typically formalized as *Markov decision processes* (MDPs) [62], given by a tuple $M = (S, A, R, T, \gamma)$, where S is a set of states; A is a set of actions, with $A(s) \subseteq A$ denoting the actions available at state s; $R(s, a, s')$ is the reward received when executing action $a \in A(s)$ in state s and transitioning to state s'; $T(s'|s, a)$ is the probability of entering state s' upon execution action a in state s; and $\gamma \in [0, 1]$ is a discount factor. A decision process is *Markov* when: 1) State s is sufficient for determining which actions are executable; i.e., $A(s)$ is only a function of s. 2) State s and action a are sufficient for determining the probability of transitioning to state s'; i.e., the distribution $T(s'|s, a)$ is conditionally independent of all other variables. 3) State s, action a, and successive state s' determine reward, so $R(s, a, s')$ depends on no other variables. The state space of a task must therefore support Markov reward (R), transition (T), and available action (A) functions.

We focus on the multi-task RL setting where the agent must solve several MDPs drawn from a task distribution. We model this as a fixed *ground MDP* to which absorbing goal states (with corresponding rewards) are superimposed to create new tasks. We define the ground MDP M_0 as: $M_0 = (S_0, A_0, R_0, T_0, \gamma)$, with state space S_0, action space A_0, transition function T_0, discount factor γ, and background reward function R_0. Any individual task $\tau \sim P(\tau)$ is given by a *goal state set* G_τ and an additional goal reward function R_τ that is non-zero only for transitions entering G_τ.

Action Abstraction *Temporal* or *action abstractions* are constructed over the primitive action set A_0 [14]. Although other formalizations exist [22, 49] (see Klissarov et al. [36] for a comprehensive overview), we adopt the *options framework* [64], where the agent has a collection of options O, where each $o \in O$ is a tuple (I_o, β_o, π_o). The *initiation set*, $I_o \subseteq S$, is the set of states from which o may be invoked. It is often defined as a classifier $I_o : S \to \{0, 1\}$. To refer to the availability of all options at a given state, we define $I_O : S \to \{0, 1\}^{|O|}$ to be a vector with entry i set to $I_{o_i}(s)$. Instances of the *initiation vector* are written as I. Similarly, the *termination condition*, $\beta_o : S \to [0, 1]$, is the probability of option termination when entering s. $\beta_O : S \to [0, 1]^{|O|}$ is the *termination vector*: a vector of every option's termination probability at state s. Finally, *option policy*, $\pi_o : S \to A$, controls option execution, which begins from a state in I_o, follows π_o, and terminates with probability $\beta_o(s_t)$ at each state s_t.

Replacing the primitive actions of an MDP with options results in a *semi-Markov decision process* (or SMDP) $M = (S, O, R, T, \gamma)$, where now O is a set of options, with $O(s) = \{o \in O | s \in I_o\}$ denoting the options available in state s; $R(s, o, s')$ returns the expected summed discounted reward received when executing option $o \in O(s)$ at state $s \in S$ and exiting in $s' \in S$; $T(s', t|s, o)$ describes probability of arriving in $s' \in S$, t time steps after executing $o \in O(s)$ from $s \in S$. The *expected-length* option model [3] represents these quantities separately, using transition function $T(s'|s, o)$ and discount model $\gamma(s, o) = \gamma^\tau$, where τ is the expected transition length. An *option model* for o consists of initiation set I_o, reward model R, transition model T, and discount model γ. This representation is sufficient for model-based planning [64, 57, 63] and can be substantially faster than low-level planning when options shorten the effective solution depth [60, 50]. An agent not given a collection of options must discover them itself. *Skill discovery algorithms*—an active area of research [27]—typically identify β_o and then learn the remaining components to successfully reach it. Our framework is intentionally agnostic to the origin of the agent's available options.

State Abstraction State abstraction [44] attempts to reduce the state space size by discarding irrelevant information or compressing the state space. Previous work has largely defined a state abstraction using an *abstraction function*: $\phi : S \to \overline{S}$ mapping each state in the original state space S to a state in a new abstract state space $\overline{S}$. This approach is often called state aggregation [44]. Although there has been some work on option-specific state abstractions for efficient option policy learning [32, 39, 19], the two abstraction types are typically considered separately.

3 Skill-Driven Abstract MDPs

Given a ground MDP $M = (S, A, R, T, \gamma)$, our goal is to construct an abstract MDP $\overline{M} = (\overline{S}, \overline{A}, \overline{R}, \overline{T}, \gamma)$. The core assumption behind almost all HRL is that $\overline{A}$ is a set of options defined over M but typically $\overline{S} \equiv S$, missing the opportunity to build a completely abstract MDP.

While we could attempt state and action abstraction independently and hope that the resulting MDP is well-formed, that seems unlikely to succeed, so one type of abstraction must be constructed to support the other.[2] Although the reverse is certainly possible (see Section 5), consider the properties that make a state Markov: it must support determining which actions can be executed, the distribution over successor states given an action, and the reward. *An observation space qualifies as a state space solely by virtue of supporting these functions, which all model the operations of actions,* and an abstract state space that satisfies these properties for a given option set leads to a well-formed MDP. Our approach therefore constructs the state abstraction to support the action abstraction.

3.1 The Semantics of Abstract States

One important question that must be answered when constructing an abstract state space is one of *semantics*: what relationship should hold between an abstract state and the ground MDP? Previous work categorizes state abstraction types according to which properties they ensure hold in the ground MDP [44]. Instead, we adopt a *constructive* approach [40] by defining a target computation and then designing an abstraction that supports it by construction. To support as wide a range of learning algorithms as possible, we target the Bellman equation [16]—the basis of virtually all value-function-based RL algorithms—which for evaluating policy π (over options) at state s gives:

$$\begin{aligned} V(s) &= \textstyle\sum_{\overline{a}} \pi(\overline{a}|s) \mathbb{E}\left[r + \gamma^{\tau} V(s')\right] = \sum_{\overline{a}} \pi(\overline{a}|s) \left(\mathbb{E}\left[R(s, \overline{a}, s')\right] + \mathbb{E}\left[\gamma^{\tau} V(s')\right]\right) \\ &\approx \textstyle\sum_{\overline{a}} \pi(\overline{a}|s) \left(\mathbb{E}\left[R(s, \overline{a}, s')\right] + \gamma^{\mathbb{E}[\tau]} \mathbb{E}\left[V(s')\right]\right), \end{aligned} \tag{1}$$

where expectations are over s' and τ given $(s, \overline{a})$, and the approximation is the expected-length model [3]. Notice that the last equation is primarily composed of *expectations over distributions of states*.

If we refer to a distribution over states using a bar (e.g. $\bar{s}$), the corresponding expected value and rewards are: $V(\bar{s}) = \mathbb{E}[V(s)]$, $R(s, \overline{a}, \bar{s}') = \mathbb{E}[R(s, \overline{a}, s')]$, and $R(\bar{s}, \overline{a}, \bar{s}') = \mathbb{E}[\mathbb{E}[R(s, \overline{a}, s')]]$. Using these definitions, Equation 1 is:

$$V(s) \approx \textstyle\sum_{\overline{a}} \pi(\overline{a}|s)[R(s, \overline{a}, \bar{s}') + \gamma^{\bar{\tau}} V(\bar{s}')],$$

where $\bar{\tau} = \mathbb{E}[\tau]$ and $\bar{s}' = T(s'|s, \overline{a})$. Later, we will pursue desirable properties by decomposing (or *refining*) $T(s'|s, \overline{a})$ into a mixture distribution, i.e., a weighted sum of component distributions. Figure 1 visualizes such a refinement.

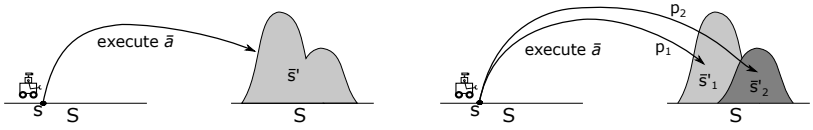


Figure 1: The outcome of executing option $\overline{a}$ from s is described by distribution over states $\bar{s}'$ (a). $\bar{s}'$ can be refined into a mixture of component distributions $\bar{s}'_1$ and $\bar{s}'_2$ with corresponding probabilities p_1 and p_2, such that $\bar{s}' = p_1 \bar{s}'_1 + p_2 \bar{s}'_2$ and $p_1 + p_2 = 1$ (b).

Refining $\bar{s}'$ into component distributions $\bar{s}'_i$, i.e., $T(s'|s, \overline{a}) = \sum_i p_i \bar{s}'_i$ where $\sum_i p_i = 1$, obtains:

$$V(s) \approx \textstyle\sum_{\overline{a}} \pi(\overline{a}|s) \sum_i p_i \left[R(s, \overline{a}, \bar{s}'_i) + \gamma^{\bar{\tau}_i} V(\bar{s}'_i)\right]. \tag{2}$$

Equation 2 is reminiscent of the Bellman equation but contains states of two types: individual states (s) on the left side and distributions over states ($\bar{s}_i$) on the right. We next extend the left side of Equation 2 to refer to a distribution of states, computing $V(\bar{s})$ for distribution $\bar{s}$, which is required to

[2] That does not mean that one process completely precedes the other in *time*; rather, one *logically* precedes the other—one type of abstraction is designed to match some property of the other.

recursively compute the right side of Equation 2. This is a strict generalization since any state can be viewed as a Dirac delta distribution. Substituting the definition of $V(\bar{s})$:

$$V(\bar{s}) \approx \mathbb{E}_{s\sim\bar{s}}\left[\sum_{\overline{a}} \pi(\overline{a}|s) \sum_i p_i[R(s,\overline{a},\bar{s}'_i) + \gamma^{\bar{\tau}_i} V(\bar{s}'_i)]\right].$$

This resemblance to the original Bellman equation suggests that we should choose abstract states $\bar{s}$ that correspond to distributions over ground states. If we choose to do so, then an abstract policy $\overline{\pi}$ is a function of $\bar{s}$ rather than s, so we can shift it out of the expectation to obtain:

$$V(\bar{s}) \approx \sum_{\overline{a}} \overline{\pi}(\overline{a}|\bar{s}) \sum_i p_i \left[R(\bar{s},\overline{a},\bar{s}'_i) + \gamma^{\bar{\tau}_i} V(\bar{s}'_i)\right]. \tag{3}$$

Equation 3 is the original Bellman equation applied to abstract states and actions, and is the sole equation necessary for computing Bellman updates. In other words, *the Bellman equation for a given set of options can therefore be naturally rewritten as a Bellman equation between abstract states, provided each abstract state refers to a distribution over ground states.* It remains to determine how to refine these state distributions so that the Markov property holds (that is, the model is an MDP), and $\overline{\pi}^*$ operates with minimal value loss over M.

3.2 Satisfying the Markov Property

The three Markov properties we seek are: 1) the abstract transition probability and discount models can be written as $T(\bar{s}'|\bar{s},\overline{a})$ and $\gamma(\bar{s},\overline{a})$; 2) the abstract reward function can be written as $R(\bar{s},\overline{a},\bar{s}')$; and 3) the options executable at an abstract state can be written as $A(\bar{s})$. The first two hold by construction but the third does not, because the distribution over states resulting from an option execution may contain states from which different options are executable. Therefore, preserving the Markov property in the abstract MDP requires that state distributions be refined (at least) into component distributions from which a consistent set of options can be executed. Figure 2 gives an example distribution and initiation set combination where this occurs and the refinement that resolves it. If the initiation sets are known exactly the component distributions do not overlap;

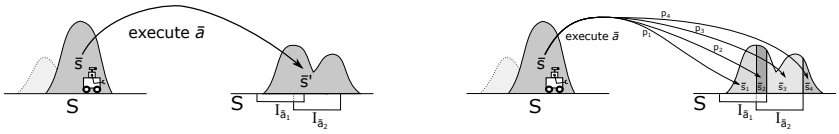


Figure 2: Option execution where the distribution over next states cannot specify which options are available next. (a) Executing $\overline{a}$ from distribution $\bar{s}$ leads to state distribution $\bar{s}'$. Some states in $\bar{s}'$ are in initiation set $I_{\overline{a}_1}$, some in $I_{\overline{a}_2}$, some in both and some neither. Therefore $\bar{s}'$ is not a Markov state as it does not specify available actions. (b) $\bar{s}'$ can be refined into distributions $\bar{s}_1$–$\bar{s}_4$ (occurring with probability p_1–p_4), each over states with the same available options—only $\overline{a}_1$ can be executed in $\bar{s}_1$, both $\overline{a}_1$ and $\overline{a}_2$ can be executed in $\bar{s}_2$, only $\overline{a}_2$ in $\bar{s}_3$, and neither in $\bar{s}_4$.

when initiation sets are uncertain (i.e., we only have a probabilistic estimate of when an option is executable), they may have overlapping support. In either case, each component distribution now represents *the distribution over states conditioned on a specific initiation set vector being observed*, so the semantics of abstract states are:

Definition 1. An **abstract state** $\bar{s}$ is a tuple (p, I), where $p(s \mid I)$ is a distribution over ground states conditioned on observed initiation vector I. We define a *grounding function* for each component: $\mathcal{G}_p(\bar{s}_i) = p_i$ and $\mathcal{G}_I(\bar{s}_i) = I_i$.

We can now form an abstract MDP $\overline{M} = \left(\overline{S}, \overline{A}, \overline{R}, \overline{T}, \bar{\gamma}\right)$ constructed over ground MDP M a follows. The actions $\overline{A}$ are a set of options over M, and the state set $\overline{S}$ is all abstract states over S: $\overline{S} = \{\bar{s} \mid \exists p, I\ \mathcal{G}_p(\bar{s}) = p, \mathcal{G}_I(\bar{s}) = I\}$. The transition function is:

$$\overline{T}(\bar{s}'|\bar{s},\overline{a}) = \begin{cases} P(I' \mid \bar{s},\overline{a}) & \text{if } \mathcal{G}_p(\bar{s}') = P(s' \mid \bar{s},\overline{a},I') \text{ and } \mathcal{G}_I(\bar{s}') = I'. \\ 0 & \text{otherwise,} \end{cases}$$

where $P(I' \mid \bar{s},\overline{a})$ is the expected probability of observing initiation vector I' after executing $\overline{a}$ from $s \sim \mathcal{G}_p(\bar{s})$, and $P(s' \mid \bar{s},\overline{a},I')$ is the expected distribution over s' after executing $\overline{a}$ from

$s \sim \mathcal{G}_p(\bar{s})$ and observing I'. The transition function expresses the probability that executing an option from abstract state $\bar{s}$ will lead to various abstract states $\bar{s}'$, each representing a distribution over states conditioned on a specific initiation vector post-execution. Similarly the discount is $\overline{\gamma}(\bar{s}, \bar{a}) = \mathbb{E}_{s \sim \mathcal{G}_p(\bar{s})}[\gamma(s, \bar{a})]$ and the reward function is: $\overline{R}(\bar{s}, \bar{a}, \bar{s}') = \mathbb{E}_{s \sim \mathcal{G}_p(\bar{s})}\left[\mathbb{E}_{s' \sim \mathcal{G}_p(\bar{s}')}[R(s, \bar{a}, s')]\right]$.

3.3 Building a Model-Preserving Abstract MDP

The above MDP is Markov by construction and supports computing the exact expected-length value of any start state distribution—because Equations 1 and 3 are equal—with two caveats: there is no obvious way to build the uncountably infinite state space, and only abstract policies $\pi(\bar{a}|\bar{s})$ are supported, when further refinement might do better. *Approximately model-preserving abstractions* [1] support computing approximately optimal policies in the multi-task setting. An abstract MDP is approximately model preserving if:

$$\int_{s'} |T(s'|s, \bar{a}) - T(s'|\bar{s}, \bar{a})|\, ds' \leq \epsilon_T \text{ and } |R(s, \bar{a}, \bar{s}') - R(\bar{s}, \bar{a}, \bar{s}')| \leq \epsilon_R,$$

for any option $\bar{a}$, pair $(\bar{s}, \bar{s}')$, ground state $s \sim \bar{s}$, and $\epsilon_R, \epsilon_T \geq 0$. Since an approximately model-preserving abstraction is also approximately value- and policy-preserving [1, 45], the optimal policy computed for an abstract MDP that obeys these conditions is also a good approximation to the optimal policy over the options in the ground MDP; indeed:

Theorem 3.1. *$\forall \bar{s}, \bar{a}$, $\pi : S \to \overline{A}$, and $s \sim \bar{s}$, if $\int_{s'} |T(s' \mid \bar{s}, \bar{a}) - T(s' \mid s, \bar{a})|\, ds' \leq \epsilon_T$ and $|R(\bar{s}, \bar{a}) - R(s, \bar{a})| \leq \epsilon_R$, then $|Q^\pi(s, \bar{a}) - Q^\pi(\bar{s}, \bar{a})| \leq \frac{\epsilon_R + \gamma \textsc{VMax} \epsilon_T}{1-\gamma}$. (Proof in the Appendix.)*

States for which this property does not hold can be refined until they do, as formalized by Algorithm 1, which first constructs an abstract MDP with abstract states for the general state distribution ($\bar{s}_{0,j}$) and every option i's outcome distribution ($\bar{s}_{i,j}$), partitioned by initiation vectors I_j. States from which a transition exceeds an error threshold are identified and iteratively refined. The abstract transition and reward functions are computed from data by taking a weighted average of samples after the abstract states are constructed. The abstraction is finite, Markov, and transition-preserving model when no such states remain and is sufficient for planning, after modification for a given *plan query* (see details in Appendix E).

Algorithm 1 Constructing and Refining an Abstract MDP

Given: transitions $\{s_i, I_i, \bar{a}_i, r_i, s_i', I_i', t_i\}, \gamma, \epsilon_t, \epsilon_r$

▷ Partition transitions by initiation vector.
$\bar{s}_{0,j} \leftarrow \{(s_i, I_i) | I_i = I_j\}, \forall I_j$
$\bar{s}_{j,k} \leftarrow \{(s_i', I_i') | \bar{a}_i = j, I_i' = I_k\}, \forall j, I_k$
$r_{j,k}, t_{j,k} \leftarrow \text{mean}(r_i, t_i | \bar{a}_i = j, I_i' = I_k)$

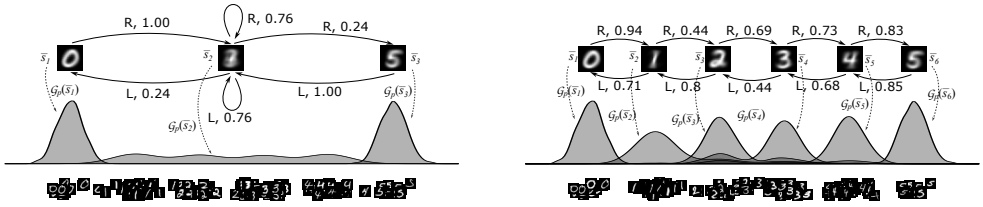
▷ Construct outgoing edges for executable actions.
for $\bar{s}, \bar{a}_i$ **where** $\bar{s}.I[i] = true$ **do**
    $\overline{T}(\bar{s}_{i,n} \mid \bar{s}, \bar{a}_i) \leftarrow (|\bar{s}_{i,n}| / \sum_m |\bar{s}_{i,m}|)$
    $\overline{R}(\bar{s}, \bar{a}, \bar{s}_{i,n}) \leftarrow r_{i,n},\ \overline{\gamma}(\bar{s}, \bar{a}, \bar{s}_{i,n}) \leftarrow \gamma^{t_{i,n}}$
end for
$\overline{S} = \{\bar{s}_{k,l}\} \forall k, l;\ \overline{A} = \{\bar{a}_i\} \forall i$

▷ Repeatedly refine states with high-error transitions.
while $\exists \bar{s}, \bar{a}, \bar{s}'$ s.t. model_error$(\bar{s}, \bar{a}, \bar{s}') \geq (\epsilon_t, \epsilon_r)$ **do**
    refine_state$(\overline{S}, \overline{A}, \overline{T}, \overline{R}, \overline{\gamma}, \bar{s})$
end while
return $\overline{M} = (\overline{S}, \overline{A}, \overline{T}, \overline{R}, \overline{\gamma})$

3.4 Examples of Learned Abstract MDPs

We demonstrate our approach using a high-dimensional chainwalk domain of length 6 (Figure 3a). The agent is equipped with actions to move left or right to adjacent states, but with 5% probability,

the subsequent state is selected uniformly at random. At each state, the agent observes a sampled 28×28 MNIST digit [42] (see Appendix A.1 for more detail).



(a) (b)

Figure 3: Learned state abstractions for the MNIST chainwalk. (a) Abstract decision process with three states prior to any refinement. Edges indicate actions with their most likely transitions. Each abstract state is sufficient to determine what actions are executable, and are depicted by their average low-level observations, while grounded samples are shown below. (b) Post-refinement abstract MDP, where the the six underlying discrete states have been recovered.

Algorithm 1 first constructs an abstraction based on the initation vectors (lines 4–13), resulting in a decision process with three abstract states (Figure 3a): the first abstract state grounds to low-level states where only the right action is available, the second to states where both actions are available, and the last to those where only left is executable. Lines 15–18 then iteratively refine these abstract states to recover a 6-state decision process that supports the Markov property (Figure 3b). Model errors are computed using a classifier two-sample test [46], while refinement is achieved by clustering. Implementation details are provided in the appendix.

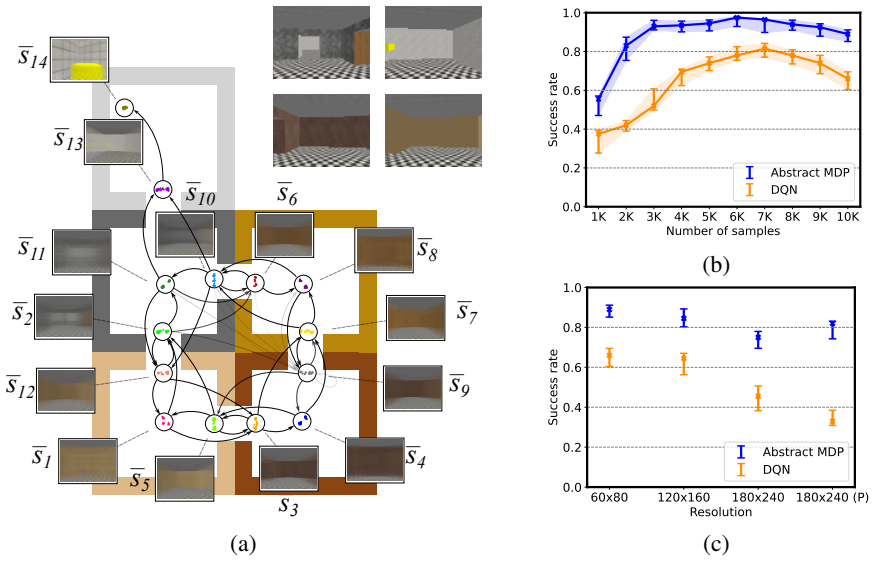
A simple Gaussian mixture model clustering algorithm is sufficient to refine abstract states in the chainwalk example, but naively clustering more complex observations can be difficult. A common approach in these cases is to first transform the observed data into a more compact form, either by using the features extracted from a pretrained neural network, or explicitly learning a latent space using reconstruction-based objectives. However, such methods do not preserve the Markov property. We therefore use Markov State Abstractions (MSAs) [6], a neural method which approximates provably sufficient conditions to recover Markov representations. A transformation ϕ that preserves the Markov property should satisfy the following conditions: the ground and abstract (1) inverse models $I^\pi(a \mid \phi(s'), \phi(s)) = I^\pi(a \mid s', s)$ and (2) their density ratios $p^\pi(\phi(s') \mid \phi(s))/p(\phi(s')) = p^\pi(s' \mid \phi(s))/p^\pi(s')$ are equal. MSAs trained with raw observations output a compressed representation that is approximately Markov, to which Algorithm 1 can be applied.

We apply the resulting neurosymbolic algorithm to a visual Miniworld [17] maze domain. The maze consists of five rooms connected by hallways, and the agent is equipped with options to navigate from the center of rooms to connecting hallways, and vice versa. The north-most room contains a gold block that the agent can approach and pickup. The walls of each room have a unique texture, and the low-level state is given by a $60 \times 80 \times 3$ image of the agent's point of view (Figure 4a, top right). We apply MSA to learn a 8-dimensional representation of the state space, and then apply Algorithm 1. Figure 4a shows the learned abstractions after the refinement, with our approach correctly recovering the underlying abstract graph structure of the environment.

Planning can then take place over this graph using any compatible planner; we ran value iteration on the abstract MDP to compute the goal-conditioned policy.

The planner successfully navigates the agent to west and then north, passing through three hallways, before collecting the gold. We further compare the planning performance of the learned abstract MDP with the goal-conditioned Deep Q-Network (DQN) [48] using the Stable Baselines 3 implementation [51].[3] Figure 4b shows that the abstract MDP achieves better performance with far fewer samples than DQN. In Figure 4c, we increased the observation resolution from 60×80 pixels to 120×160 and 180×240 pixels, while keeping the sample size fixed. As an additional complication, we added random portraits on the wall for the highest resolution—indicated with (P). Although these changes increase both the dimensionality and the complexity of the input space, the underlying decision

[3]Most RL libraries assume that every action is available in every state. To make a fair comparison with SB3's DQN implementation, we used the same initiation vector for all ground states during abstract MDP construction.



(a) (b) (c)

Figure 4: Application of Algorithm 1 with MSA to the visual maze domain. The first abstract MDP is constructed by aggregating states with the same initiation vector. Abstract states are then iteratively refined until there is no improvement on the transition error. (a) Learned abstract MDP consisting of 14 abstract states after the refinement process. Each abstract state is assigned to its own color, while insets show the average ground state at various locations in the domain. The planning performance of methods trained with varying (b) number of samples and (c) resolutions averaged over 20 runs. Shaded regions denote two standard errors around the mean, and the median is marked with a cross.

process remains the same—the correct abstract model should be invariant to the resolution and to the distractor portraits. The results in Figure 4c show that the abstract MDP's performance is not affected by these modifications, whereas the DQN's performance steadily declines with the increasing input complexity. This supports the objective of abstraction: to retain only the information that is necessary for effective decision-making.

4 Factored Skills Generate Factored MDPs

The above formulation assumes that option execution always changes every element of the low-level state vector. In practice, agents often have options that change some state variables but not others. Such *factored* options are common in high-dimensional domains like robotics, and we now show that factored options with a subgoal property lead to factored abstract states. Since abstract states ground to distributions over low-level states, we formalize a factored option as one where execution changes the distribution over some state variables, but leaves the distribution over others unchanged (after marginalizing out the modified variables). Formally:

Definition 2. An option o is factored with *mask* m from distribution $\overline{s} = P(s[m], s[\overline{m}] \mid I_O)$ if $T(s'|s \sim \overline{s}, o, I'_O) = P(s'[m]|s \sim \overline{s}, o, I'_O) \times P(s[\overline{m}]|I_O)$, where m are the masked state variables, $\overline{m}$ are those *not* in m, and $s[m]$ and $s[\overline{m}]$ are the corresponding components of s.

Executing factored option o from $\overline{s}$ with mask m differs in its effects on the masked and unchanged state variables. The new distribution on masked variables depends on the start state (drawn from $\overline{s}$) and post-execution initiation vector I'_O. The distribution over the unchanged variable is simply $\overline{s}$ with masked variables marginalized out. These two distributions are independent; the variables in $\overline{m}$ are unchanged by o so s' and I'_O do not inform their distribution, while those in m already depend on s, so $s[\overline{m}]$ is redundant. This definition can be naturally extended to multiple possible factored outcomes, each with their own mask m_i:

Definition 3. An *outcome* is a tuple $\omega = (m, I'_O, P(s[m]))$, where m is a mask, I'_O is an observed initiation vector, and $P(s[m])$ is a distribution over the state variables in the mask. Outcome ω_i results from executing option o from distribution $\bar{s}$ when: only the state variables in m change; the

agent observes subsequent initiation vector I'_O; and $P(s[m])$ describes the subsequent distribution over the modified state variables, i.e., $P(s'[m] \mid o, \bar{s}, I'_O)$.

The condition supporting a model-preserving discrete abstract state space also generalizes naturally to factors—that the distribution over modified variables depends on $\overline{s}$ but not low-level state $s \sim \overline{s}$:

Definition 4. A factored option o with mask m is a *factored subgoal option* from distribution $\overline{s} = P(s \mid I_O)$ if: $P(s'[m] \mid s \sim \overline{s}, o, I'_O) = P(s'[m] \mid \overline{s}, o, I'_O)$.

4.1 Defining a Factored Abstract MDP

The abstract MDP for the factored case must be built out of the masked distributions $P_i(s'[m_i] \mid \overline{s}, o, I'_{Oi})$ that result from factored subgoal option execution. These distributions are only defined over a *subset* m_i of state variables in the ground MDP, and fulfilling any of the Markov properties requires representing a distribution over *all* state variables. We must therefore model how each factored subgoal option modifies a subset of the low-level state variables, while representing a distribution over all of them. We achieve this by grouping the low-level state variables into factors, where an abstract state assigns a factor value to every factor and grounds to a distribution over all low-level state variables, but option execution changes only some of these values.

Definition 5. An *atomic factor* $f \subseteq S$ is a group of low-level state variables that are either unmodified by option execution or are all modified together, i.e., $(m_i \cap f \neq \emptyset) \implies f \subseteq m_i$, for all outcome masks m_i. Let $F = \{f_1, ..., f_n\}$ be the set of all such factors for a low-level MDP.

An outcome "sets" some of these factors—those inside its mask—to an abstract value representing its masked distribution. We therefore represent each factor as an abstract state variable that can be assigned one of a set of *factor values* representing an outcome ω_i that includes that factor.

Definition 6. A factor f_j has a *factor value* $e_{j,i}$ for each outcome ω_i for which $f_j \subseteq m_i$, i.e., for each outcome that changes the variables in the factor.

An abstract state is then an assignment of values to every factor, representing a distribution over every low-level state variable and an observed initiation vector.

Definition 7. Given a ground MDP with factored subgoal options and n factors, an abstract state $\overline{s}$ is an assignment of exactly one factor value $e_j \in E_j$ to each factor f_j: $\overline{s} = [e_1, ..., e_n]$, and an initiation vector $\bar{s}_g = I_O$. The abstract state space $\overline{S}$ is the set of all such assignments.

Transitions in the abstract MDP set the factors overlapping the relevant outcome's mask to the factor values representing its effect; other factor values remain unchanged. The remaining decision process construction—which satisfies all three Markov properties—follows straightforwardly, as does the learning algorithm (both are detailed in the Appendix). Abstract states are grounded as:

Definition 8. The state distribution grounding of factored abstract state $\bar{s}$ is the product of the subgoal distributions for each outcome with effects present in $\bar{s}$, after the factors belonging to each outcome absent in $\bar{s}$ have been integrated out. Formally:

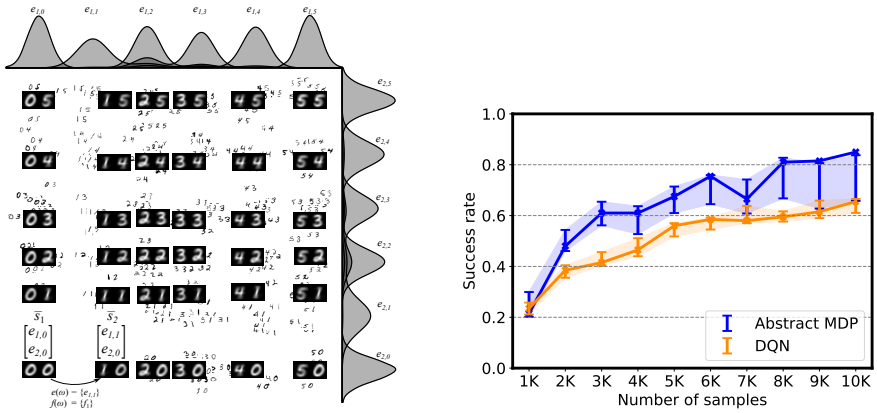
$$\mathcal{G}_p(\bar{s}) = \Pi_{\omega_i \in \omega(\bar{s})} \left[\int \cdots \int_{\bar{e}(\omega_i, \bar{s})} P_i(s[m_i]) d \cdots d_{\bar{e}(\omega_i, \bar{s})} \right],$$

where $\omega(\bar{s}) = \{\omega_i | \exists j \ \bar{s}[j] = e_j(\omega_i)\}$ is the collection of outcomes with an effect value in $\bar{s}$, and $\bar{e}(\omega_i, \bar{s}) = \{e_j(\omega_i) | \bar{s}[j] \neq e_j(\omega_i)\}$ are all of ω_i's effect values that are *not* present in $\bar{s}$. The initiation set grounding is simply the available initiation vector: $\mathcal{G}_I(\bar{s}) = \bar{s}_g$.

Theorem 4.1. *Given abstract factored option $\overline{a}$ and abstract state $\overline{s}$ with grounding $\mathcal{G}_p(\overline{s})$, $T(\overline{s}' | \overline{s}, \overline{a})$ grounds to the distribution over states resulting from executing $\overline{a}$ from a state drawn from $\mathcal{G}_p(\overline{s})$. (Proof in the Appendix.)*

4.2 Examples of Learned Factored Abstract MDPs

Figure 5a shows the resulting factored abstract MDP on a gridwalk domain, an extension of chainwalk where there are two integers representing the underlying state in each axis, and actions are the cardinal directions that change the state only in one axis. As a result, the learned abstract states, $\{\bar{s}_1, \ldots, \bar{s}_{36}\}$ are now defined as the assignment of a factor value to each factor, $\bar{s} = [e_{1,i}, e_{2,j}]$, over six different



(a) (b)

Figure 5: Learned factored abstract MDP on MNIST gridwalk. (a) Abstract states correspond to factored distributions over two factors. Observations are a concatenation of two 28×28 sized digit images, representing the state in each axis. (b) Planning performance for varying number of samples. Shaded regions denote two standard errors around the mean, and the median is marked with a cross.

values for each factor. Similar to the previous experiment, we compare the planning performance with the goal-conditioned DQN with hindsight experience replay [8] for varying number of samples. We trained an MSA network with 2-dimensional representations on which we can apply Algorithm 1. We added an auxiliary loss to the original MSA objective to minimize the mutual information between hidden units, estimated separately with InfoNCE [67], which allows us to approximate the hidden units as factors. Figure 5b shows that the abstract MDP performs better with fewer samples when compared with DQN. The high variance in abstract MDP performance is due to suboptimal factorization of the MSA embedding space in some runs, which leads to poor refinements. Note that the median values are close to the upper bound of the two standard error bars, suggesting that the factored refinement works well only when the embedding space is properly factored. As such, learning a Markov representation that is factored with respect to the abstract actions would directly improve the task performance [52].

We next apply the algorithm to the challenging Atari game *Montezuma's Revenge* [15]. The state space is given by the annotated RAM states [7] corresponding to 14 factors over 16 variables, while the agent is equipped with a set of factored subgoal options, such as climbing down ladders (see the appendix for the full list), which change only a subset of the RAM variables. We use an expert policy that completes the first room 10% of the time, and otherwise deviates from the plan and executes random actions. This privileged process can be replaced by an exploration module that visits abstract states with fewer samples (or higher transition errors) to expand the frontier of the abstract MDP by further refinement, which we leave as future work. Figure 6 shows visualizations for some of the factored distributions arising from the learned abstraction. After the final refinement, there are 116 factor values over 14 factors that resulted in 1816 abstract states. We then apply value iteration to the resulting abstraction, with the agent learning a policy capable of collecting the key and exiting the room. Figure 6 shows the trajectory of the policy's execution.

5 Related Work

Our work builds on constructive approaches in robotics [40, 29] for learning classical planning representations. That work does not result in Markov representations and can only reason about open-loop, straight-line plans. Others have learned object-centric classical planning representations from pixels using deep networks [65, 66, 5, 10] but without any formal guarantees, or using program synthesis [30, 25, 58], which faces scaling challenges. Earlier work in hierarchical RL [20, 34, 22] employed abstract states with hierarchical actions, but they primarily focused on learning policies within fixed hierarchies rather than learning the abstractions themselves.

Aggregation-based state abstractions can be organized into a hierarchy based on whether they preserve specific properties ranging from optimal policies to rewards and transition dynamics for all policies

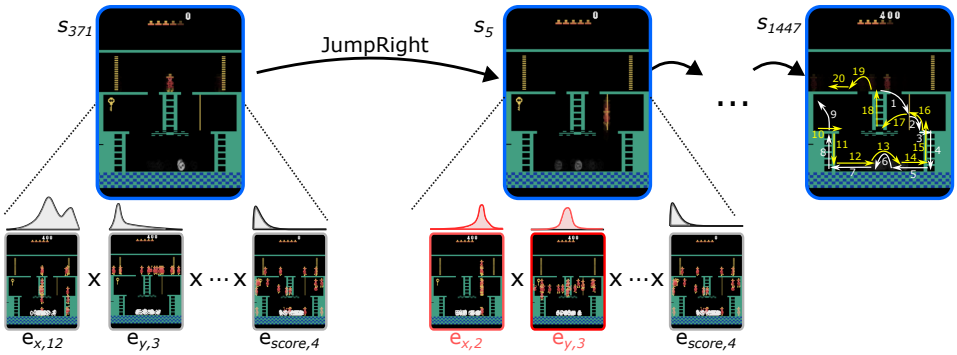


Figure 6: Examples of factored distributions on *Montezuma's Revenge*. Each abstract state (in blue) is an assignment of one factor value (in gray). A factored subgoal option changes some of these factor assignments while leaving the others unmodified. In this example, `JumpRight` action changes factors that correspond to the agent's x and y location and the skull's x location. The execution of the simulated plan that maximizes the reward by applying value iteration is shown on the right.

[44]. The strictest class of abstractions is commonly called model-preservation or bisimulation [21]. Abel [2] provides bounds for $(\phi, \mathcal{O}_\phi)$ state-action abstractions, one of which is similar to Theorem 3.1. Our main difference from these previous works is that we define abstract states as distributions over ground states, as opposed to sets. Even though soft state-aggregation-based methods [59, 9] allow defining distributions, because their abstraction is a function from the ground state to the abstract, supports of different abstract states cannot overlap. Furthermore, these abstractions are not tied to any specific semantics of the ground MDP, and thus, can arbitrarily fail to take into account the shape of the ground distribution. Defining abstract states as distributions over ground states—as the primary structure—supports the recursive computation of the Bellman equation over the abstract policy. State groundings can overlap and have different density functions. This difference in abstract state semantics is a fundamentally different direction in state abstraction.

There is a rich line of work that uses neural networks to learn state abstractions independently from action abstraction, typically by converting intuitively desirable properties into loss functions. These include compression [47], next-state prediction [68], inverse dynamics modeling [18], successor features [13], robotic priors [31], contrastive learning [67], or combinations of these [4, 56, 26]. Allen [6] surveys these techniques and characterizes the specific combination that guarantees the Markov property. The result is MSA, which is compressed but not abstract or skill-driven.

A few works have constructed abstract actions based on abstract states, typically partitioning low-level states using graph-based clustering and constructing options to transition between adjacent clusters [37, 69, 41, 33, 61, 55, 54, 24]. These approaches assume that options can always be constructed between adjacent generated abstract states. That assumption fails in many tasks because options are subject to environment dynamics whereas state abstractions are not. Finally, two recent approaches perform skill-driven state abstraction. Bagaria et al. [12, 11] construct options to form a connected graph and uses the nodes as abstract states. Shah et al. [53] use an option's value function evaluated at a low-level state as an abstract state variable. Neither approach guarantees the Markov property or value preservation.

6 Conclusion

Hierarchical RL has always focused on abstract actions to avoid the complexity of generating very long sequences of low-level actions. But fully realizing the promise of hierarchy requires *combining* state and action abstractions, to reformulate a complex agent-level MDP into a entirely new, and potentially much simpler, abstract MDP. Such task-level abstraction is a promising approach to realizing generally-intelligent agents [23, 38], which must necessarily have very complex sensorimotor spaces, and mimics the human ability to form and exploit abstract task representations [28]. Our work combines principled theoretical approaches with the power of neural networks to provide a powerful framework for constructing abstract MDPs.

Acknowledgments

This research was supported by the ONR under REPRISM MURI N00014-24-1-2603 and grant N00014-22-1-2592, and in part by the NSF Graduate Research Fellowship (Grant No. 2040433). Cameron Allen was supported by a gift from Open Philanthropy to the Center for Human-Compatible AI at UC Berkeley, and an AI2050 Senior Fellowship for Stuart Russell from the Schmidt Fund for Strategic Innovation. Part of this research was conducted using computational resources and services at the Center for Computation and Visualization, Brown University.

References

[1] D. Abel, D.E. Hershkowitz, and M.L. Littman. Near optimal behavior via approximate state abstraction. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2915–2923, 2016.

[2] D. Abel, N. Umbanhowar, K. Khetarpal, D. Arumugam, D. Precup, and M. Littman. Value preserving state-action abstractions. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, pages 1639–1650, 2020.

[3] D. Abel, J. Winder, M. desJardins, and M.L. Littman. The expected-length model of options. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 1951–1958, 2019.

[4] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems 29*, pages 5074–5082, 2016.

[5] A. Ahmetoglu, M.Y. Seker, J. Piater, E. Oztop, and E. Ugur. DeepSym: Deep symbol generation and rule learning for planning from unsupervised robot interaction. *Journal of Artificial Intelligence Research*, 75:709–745, 2022.

[6] C. Allen, N. Parikh, O. Gottesman, and G.D. Konidaris. Learning Markov state abstractions for deep reinforcement learning. In *Advances in Neural Information Processing Systems 34*, pages 8229–8241, 2021.

[7] A. Anand, E. Racah, S. Ozair, Y. Bengio, M-A. Côté, and R.D. Hjelm. Unsupervised state representation learning in Atari. In *Advances in Neural Information Processing Systems 32*, pages 8769–8782, 2019.

[8] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[9] K. Asadi, D. Abel, and M.L Littman. Learning state abstractions for transfer in continuous control. *arXiv:2002.05518*, 2020.

[10] M. Asai, H. Kajino, A. Fukunaga, and C. Muise. Classical planning in deep latent space. *Journal of Artificial Intelligence Research*, 74:1599–1686, 2022.

[11] A. Bagaria, A. de Mello Koch, R. Rodriguez-Sanchez, S. Lobel, and G. Konidaris. Intrinsically motivated discovery of temporally abstract graph-based models of the world. *Reinforcement Learning Journal*, 6:2115–2134, 2025.

[12] A. Bagaria, J. Senthil, and G.D. Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *Proceedings of the Thirty-Eighth International Conference on Machine Learning*, pages 521–531, 2021.

[13] A. Barreto, W. Dabney, R. Munos, J.J. Hunt, T. Schaul, H. van Hasselt, and D. Silver. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems 30*, pages 4055–4065, 2017.

[14] A.G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13:41–77, 2003.

[15] M.G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[16] R. Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[17] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P.S. Castro, and J. Terry. Minigrid & Miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *arXiv:2306.13831*, 2023.

[18] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv:1610.03518*, 2016.

[19] L.C. Cobo, K. Subramanian, C.L. Isbell, A.D. Lanterman, and A.L. Thomaz. Abstraction from demonstration for efficient reinforcement learning in high-dimensional domains. *Artificial Intelligence*, 216:103–128, 2014.

[20] P. Dayan and G.E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems V*, 1992.

[21] T. Dean and R. Givan. Model minimization in Markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 106–111, 1997.

[22] T.G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[23] S. Doncieux, D. Filliat, N. Díaz-Rodríguez, T. Hospedales, R. Duro, A. Coninx, D.M. Roijers, B. Girard, N. Perrin, and O. Sigaud. Open-ended learning: a conceptual framework based on representational redescription. *Frontiers in Neurorobotics*, 12:59, 2018.

[24] J.B. Evans and Ö. Şimşek. Creating multi-level skill hierarchies in reinforcement learning. In *Advances in Neural Information Processing Systems 36*, pages 48472–48484, 2023.

[25] C.R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L.P. Kaelbling, and T. Lozano-Perez. Integrated task and motion planning. *Annual Review of control, Robotics, and Autonomous Systems*, 4:265–293, 2021.

[26] D. Ha and J. Schmidhuber. World models. *arXiv:1803.10122*, 2018.

[27] B. Hengst. Hierarchical approaches. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning*, Adaptation, Learning, and Optimization, chapter 12, pages 293–323. Springer Berlin Heidelberg, 2012.

[28] M.K. Ho, D. Abel, C.G. Correa, M.L. Littman, J.D. Cohen, and T.L. Griffiths. People construct simplified mental representations to plan. *Nature*, 606(7912):129–136, 2022.

[29] S. James, B. Rosman, and G.D. Konidaris. Autonomous learning of object-centric abstractions for high-level planning. In *Proceedings of the Tenth International Conference on Learning Representations*, 2022.

[30] N. Jetchev, T. Lang, and M. Toussaint. Learning grounded relational symbols from continuous data for abstract reasoning. In *Proceedings of the 2013 ICRA Workshop on Autonomous Learning*, 2013.

[31] R. Jonschkowski and O. Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39:407–428, 2015.

[32] A. Jonsson and A.G. Barto. Automated state abstraction for options using the U-Tree algorithm. In *Advances in Neural Information Processing Systems 13*, pages 1054–1060, 2001.

[33] K. Jothimurugan, O. Bastani, and R. Alur. Abstract value iteration for hierarchical reinforcement learning. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, pages 1162–1170, 2021.

[34] L.P. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 167–173, 1993.

[35] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232, 2002.

[36] M. Klissarov, A. Bagaria, Z. Luo, G. Konidaris, D. Precup, and M.C. Machado. Discovering temporal structure: An overview of hierarchical reinforcement learning. *arXiv preprint arXiv:2506.14045*, 2025.

[37] M.J. Kochenderfer. *Adaptive Modelling and Planning for Learning Intelligent Behaviour*. PhD thesis, University of Edinburgh, 2006.

[38] G. Konidaris. On the necessity of abstraction. *Current Opinion in Behavioral Sciences*, 29:1–7, 2019.

[39] G.D. Konidaris and A.G. Barto. Efficient skill learning using abstraction selection. In *Proceedings of the Twenty First International Joint Conference on Artificial Intelligence*, pages 1107–1112, July 2009.

[40] G.D. Konidaris, L.P. Kaelbling, and T. Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, January 2018.

[41] A.S. Lakshminarayanan, R. Krishnamurthy, P. Kumar, and B. Ravindran. Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv:1605.05359*, 2020.

[42] Y. LeCun. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`, 1998.

[43] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[44] L. Li, T.J. Walsh, and M.L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006.

[45] S. Lobel and R. Parr. An optimal tightness bound for the simulation lemma. *Reinforcement Learning Journal*, 2:785–797, 2024.

[46] D. Lopez-Paz and M. Oquab. Revisiting classifier two-sample tests. In *Proceedings of the Fifth International Conference on Learning Representations*, 2017.

[47] J. Mattner, S. Lange, and M. Riedmiller. Learn to swing up and balance a real pole based on raw visual input data. In *Proceedings of the 19th International Conference on Neural Information Processing*, pages 126–133, 2012.

[48] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[49] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*, pages 1043–1049, 1997.

[50] E.J. Powley, D. Whitehouse, and P.I. Cowling. Monte Carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem. In *Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games*, pages 234–241, 2012.

[51] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[52] R. Rodriguez-Sanchez, C. Allen, and G. Konidaris. From pixels to factors: Learning independently controllable state variables for reinforcement learning. *arXiv preprint arXiv:2510.02484*, 2025.

[53] D. Shah, P. Xu, Y. Lu, T. Xiao, A.T. Toshev, S. Levine, and B. Ichter. Value function spaces: Skill-centric state abstractions for long-horizon reasoning. In *Proceedings of the Tenth International Conference on Learning Representations*, 2022.

[54] N. Shah, J. Nagpal, and S. Srivastava. From real world to logic and back: Learning generalizable relational concepts for long horizon robot planning. In *Proceedings of the 9th Annual Conference on Robot Learning*, 2025.

[55] N. Shah and S. Srivastava. Using deep learning to bootstrap abstractions for hierarchical robot planning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, page 1183–1191, 2022.

[56] E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv:1612.07307*, 2016.

[57] D. Silver and K. Ciosek. Compositional planning using optimal option models. In *Proceedings of the Twenty-Ninth International Conference on Machine Learning*, pages 1267–1274, 2012.

[58] T. Silver, R. Chitnis, N. Kumar, W. McCLinton, T. Lozano-Perez, L.P. Kaelbling, and J. Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the 2023 AAAI Conference on Artificial Intelligence*, pages 12120–12129, 2023.

[59] S. Singh, T. Jaakkola, and M. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems 7*, pages 361–368, 1994.

[60] J. Sorg and S. Singh. Linear options. In *Proceedings of the Ninth International Conference on Autonomous Systems and Multiagent Systems*, pages 31–38, 2010.

[61] L. Steccanella, S. Totaro, and A. Jonsson. Hierarchical representation learning for Markov decision processes. In *Proceedings of the IJCAI 2021 Workshop on Generalization in Planning*, 2021.

[62] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[63] R.S. Sutton, M.C. Machado, G.Z. Holland, D. Szepesvari, F. Timbers, B. Tanner, and A. White. Reward-respecting subtasks for model-based reinforcement learning. *Artificial Intelligence*, 324:104001, 2023.

[64] R.S. Sutton, D. Precup, and S.P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[65] E. Ugur and J. Piater. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2627–2633. IEEE, 2015.

[66] E. Ugur and J. Piater. Refining discovered symbols with multi-step interaction experience. In *Proceedings of the 15th IEEE-RAS International Conference on Humanoid Robots*, pages 1007–1012, 2015.

[67] A. van den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv:1807.03748*, 2018.

[68] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems 28*, pages 2746–2754, 2015.

[69] A. Zhang, S. Sukhbaatar, A. Lerer, A. Szlam, and R. Fergus. Composable planning with attributes. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5842–5851, 2018.

A Environment Details

A.1 Visual Chainwalk

The visual chainwalk domain consists of six underlying states. Unlike the standard environment, in which the state representation of the agent is a single integer, here the state is a visual image representation of the integer, ranging from 0 to 5. At a given state, the agent receives a 28×28 greyscale image sampled from the labelled set of MNIST images [42]. At each step, a new image is sampled uniformly at random and flattened into a 784 dimensional vector.

The agent has two options (which are equivalent to its low-level actions, owing to the simplicity of the domain): moving left and right. These options are executable everywhere, except at boundary states where the agent can only move right or left appropriately. With probability 0.95, option execution is deterministic and the agent transitions to the adjacent state. However, with probability 0.05, the next state is sampled uniformly at random. For each action, the agent receives a reward of 1 if it arrives at the goal state, and 0 otherwise.

A.2 Visual Maze

The visual maze domain is built in the Miniworld [17] framework and consists of five rooms with hallways connecting the rooms. Figure 7 illustrates the layout of the maze from a top down perspective, along with the $60 \times 80 \times 3$ image of the agent's point of view, which represents the state.

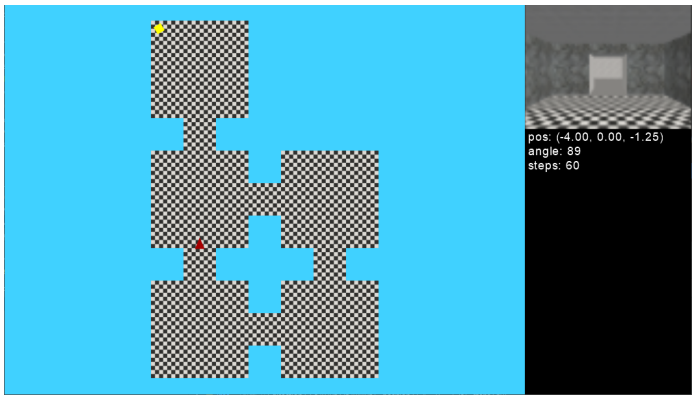


Figure 7: A view of the vault environment, showing the layout of the five rooms and the connecting hallways from a top-down perspective. The agent is represented by a red triangle, and there is a gold block in the north-most room. The state is given by the agent's egocentric view (top right), while additional information such as its exact location is also specified. Note that the state space is only the egocentric pixel observation; no other privileged information is provided to the agent.

Each room has walls with different textures (cardboard, brick, wood, rock and tile) to ensure that the egocentric observation is still Markov. The agent is equipped with five noisy options:

- "Walk to center". This is only executable when the agent is standing in a room's hallway. The option terminates when the agent is in the center of the room.
- "Walk to clockwise hallway". This is executable everywhere, except when the agent is in the north-most room. The option terminates when the agent reaches the hallway closest in the clockwise direction.
- "Walk to anticlockwise hallway". This is executable everywhere, except when the agent is in the north-most room. The option terminates when the agent reaches the hallway closest in the anticlockwise direction.
- "Walk to north hallway". This is only executable when the agent is standing in the center of the room connected to the north-most room. The option terminates when the agent reaches the hallway leading to the room with the gold block.
- "Pickup gold". This is only executable when the agent is standing in the north-most room. The option terminates when the agent has walked to the gold block and picked it up.

The agent receives a constant negative reward for each step taken in the domain, and an episode terminates when the gold block is collected.

A.3 Visual Gridworld

The visual gridworld is an extension to the chainwalk domain. Here, the state is represented by two MNIST digits, and the agent's actions (options) are each of the cardinal directions (except at boundary states). The two digits represent the underlying xy-coordinate on the grid; when the agent transitions to a new state, the image of the unchanged coordinate remains fixed. As such, the state space is of dimension 1568 with are two factors: one for state variables with indices $[0, \ldots, 783]$, and the other for indices $[784, \ldots, 1567]$.

A.4 *Montezuma's Revenge*

We use the Atari video game *Montezuma's Revenge* as an example of a challenging RL domain. Here, the state space is represented by the underlying 16 bytes that make up the subset of the RAM state that controls the game at a particular time [7]. The domain is otherwise standard, except we equip the agent with the following set of options:

- `RunLeftSkill`: the agent runs left until it reaches the end of a platform or wall.
- `RunRightSkill`: the agent runs right until it reaches the end of a platform or wall.
- `ClimbDownRopeSkill`: the agent climbs down the rope until it reaches the ground.
- `ClimbUpRopeSkill`: the agent climbs up the rope until it reaches the ground.
- `DropFromRopeSkill`: the agent jumps off the rope to the ground.
- `ClimbDownLadderSkill`: the agent climbs down the ladder until it reaches the ground.
- `ClimbUpLadderSkill`: the agent climbs up the ladder until it reaches the ground.
- `JumpInPlaceSkill`: the agent jumps vertically.
- `JumpLeftSkill`: the agent jumps left.
- `JumpRightSkill`: the agent jumps right.
- `WaitForSkullMovingTowards`: the agent stands in place until the skull starts moving towards it.
- `WaitForSkullMovingAway`: the agent stands in place until the skull starts moving away from it.
- `WaitForJumpSkullMovingTowards`: the agent waits in place and the jumps when the skull is moving towards it.
- `WaitForJumpSkullMovingAway`: the agent waits in place and the jumps when the skull is moving away from it.
- `PassToLeftOfJumpSkull`: the agent moves left away from the skull.
- `PassToRightOfJumpSkull`: the agent moves right away from the skull.
- `ChargeEnemyLeft`: the agent runs left towards the skull.
- `ChargeEnemyRight`: the agent runs right towards the skull.

B Algorithms for Computing the Model Error and Refinement

B.1 Using Classifier Two-Sample Test

We compute a surrogate value for ϵ_T by using a classifier two-sample test (C2ST) [46], which is a popular metric to evaluate generative models that produce distributions over high-dimensional vectors such as natural images. Here, we essentially want to evaluate the discrepancy between $T(s' \mid s, \bar{a})$ and $T(s' \mid \bar{s}, \bar{a})$, which is equivalent to the discrepancy between the joint probability $p(s', s)$ and the product of marginals $p(s')p(s)$ under the expectation of $\bar{s}$. We fit a k-nearest neighbor classifier with (s, s') as positive examples and their shuffled version $(\tilde{s}, s')$ as the negative ones. We measure

the deviation from 50% accuracy, at which s and s' are independent from each other and that the classifier cannot differentiate (s, s') from $(\tilde{s}, s')$. We summarize this procedure in Algorithm 2. During refinement, we sum ϵ_T values of all abstract states to come up with a cumulative transition error metric $\epsilon_{T_{\text{cum}}}$ for the whole abstract state space. We then compute $\epsilon_{T_{\text{cum}}}$ m times and make a t-test to measure the significance of the refinement and accept those that are below a certain value.

Algorithm 2 Compute State Error

Given: $\bar{s}$, transition tuples $\mathcal{D} = \{s, \bar{a}, r, s'\}$ corresponding to $\bar{s}$, factor f_i, number of tests n, number of nearest neighbors k

$\epsilon_T, \epsilon_R \leftarrow 0$
for each available option $\bar{a}_i$ **in** $\bar{s}$ **do**
  $(s_a, s'_a, r) \leftarrow$ filter_transitions_by_option($\mathcal{D}, \bar{a}_i$)
  $m \leftarrow f_i$.variables
  $\mathcal{H}_0 \leftarrow \{\}$
  $\mathcal{H}_a \leftarrow \{\}$
  for $i = 1$ **to** n **do**
    $x_{\text{ground}} \leftarrow (s_a, s'_a[m])$
    $x_{\text{abstract}} \leftarrow (s \sim \bar{s}, s'_a[m])$
    $x_0 \leftarrow (\text{shuffle}(s_a), s'_a[m])$
    $\mathcal{H}_0 \leftarrow \mathcal{H}_0 \cup \text{knn\_accuracy}(x_{\text{ground}}, x_0, k)$
    $\mathcal{H}_a \leftarrow \mathcal{H}_a \cup \text{knn\_accuracy}(x_{\text{abstract}}, x_0, k)$
  end for
  $\mu_0, \sigma_0 \leftarrow$ compute_statistics($\mathcal{H}_0$)
  $\mu_a, \sigma_a \leftarrow$ compute_statistics($\mathcal{H}_a$)
  $\epsilon_T \leftarrow \epsilon_T + p * (-\log(\text{ttest\_}p\text{-value}(\mu_0, \sigma_0, \mu_a, \sigma_a, n))$
  $\epsilon_R \leftarrow \epsilon_R + p * \text{var}(r)$
end for
return ϵ_T, ϵ_R

B.2 Refinement

We wish to decompose an abstract state into smaller components, and then recompute the appropriate transition dynamics for each of them (both for incoming and outgoing transitions). Since an abstract state represents a distribution over ground states, there are combinatorially many ways the state could be refined. In practice, we adopt a clustering approach, with each resulting cluster representing a component distribution. We find that expectation-maximization on a Gaussian mixture model with two components is sufficient for the domains presented here, but the refinement process is agnostic to this. Earlier version of this work used k-means with $k = 2$ that have similar results. We later adopted GMMs to accommodate overlapping supports. Algorithm 3 illustrates the procedure for refining a given abstract state.

Algorithm 3 refine_state

Given: $\overline{S}, \overline{A}, \overline{T}, \overline{R}, \overline{\gamma}, \bar{s}$, transition tuples $\mathcal{D}$, clustering algorithm $\mathcal{C}$

$\{s, \bar{a}, r, s'\} \leftarrow$ filter_transitions_by_abstract_state($\mathcal{D}, \bar{s}$)
$\mathcal{C}$.fit(s)
labels$_s \leftarrow \mathcal{C}$.predict($s$)
labels$_{s'} \leftarrow \mathcal{C}$.predict($s'$)
Create new abstract states $\overline{S}_{\text{new}}$ based on unique labels from labels$_s$ and labels$_{s'}$
Assign each transition $(s, \bar{a}, r, s')$ to the corresponding abstract states in $\overline{S}_{\text{new}}$
$\overline{S} \leftarrow \overline{S} \cup \overline{S}_{\text{new}} \setminus \{\bar{s}\}$
Recompute abstract transitions $\overline{T}$ based on updated transitions

B.3 Using MSA Density Ratio Loss

We used InfoNCE [67] objective to approximate the density ratio objective in MSA. This objective pushes the model to create a contrast between real transition tuples (s, s') and their shuffled version $(s, \tilde{s}')$. Since this objective provides a lower bound to the mutual information $I(s, s')$, it can be directly used as a proxy for detecting states with high mutual information. Such states can be selected as candidates for refinement.

C Proof for Theorem 3.1

Let abstract states $\bar{s}$ represent distributions over ground states, defined by $\Pr(s|\bar{s})$. Let $\bar{T}(\bar{s}'|\bar{s}, \bar{a})$ and $\bar{R}(\bar{s}, \bar{a})$ be a transition and reward functions defined over abstract states. Thus, we define $T(s'|\bar{s}, \bar{a}) = \sum_{\bar{s}'} \bar{T}(\bar{s}'|\bar{s}, \bar{a}) \Pr(s'|\bar{s}')$, and similarly define $\pi(\bar{a}|\bar{s}) = \int_s \Pr(s|\bar{s})\pi(\bar{a}|s)$. Additionally let $R(s, \bar{a}) \geq 0$. VMAX represents the maximum value obtained in either the ground or abstract MDP: $\text{VMAX} = \max(\max_s V(s), \max_{\bar{s}} V(\bar{s}))$. For conciseness, we define $T^{s,\bar{a},s'} = T(s'|s, \bar{a})$.

Theorem C.1. *For all $\bar{s}$, for any $\bar{a}$, for any policy $\pi : S \to \bar{A}$, and any $s \in \{x \in S : P(x \sim \bar{s}) > 0\}$, if*

$$\int_{s'} |T(s' \mid \bar{s}, \bar{a}) - T(s' \mid s, \bar{a})| \, ds' \leq \epsilon_T, \tag{4}$$

$$|\bar{R}(\bar{s}, \bar{a}) - R(s, \bar{a})| \leq \epsilon_R, \tag{5}$$

then

$$|Q^\pi(s, \bar{a}) - Q^\pi(\bar{s}, \bar{a})| \leq \frac{\epsilon_R + \gamma \text{VMAX} \epsilon_T}{1 - \gamma}. \tag{6}$$

Theorem 3.1. This result is closely related to the *simulation lemma* [45, 35], generalized to apply to abstract state and action spaces. Without loss of generality, we present the proof for the case of $Q^\pi(s, \bar{a}) \geq Q^\pi(\bar{s}, \bar{a})$. The same arguments hold for $Q^\pi(s, \bar{a}) \leq Q^\pi(\bar{s}, \bar{a})$. We begin by proving that a bound on Q differences implies the same bound on V differences, in expectation.

Lemma C.2. *Let Δ_Q be the maximum discrepancy for any $\bar{s}, \bar{a}$, and any $s \in \{x \in S : P(x \sim \bar{s}) > 0\}$:*

$$\Delta_Q = \max_{\bar{s}, o, s \sim \bar{s}} Q^\pi(s, \bar{a}) - Q^\pi(\bar{s}, \bar{a}) \tag{7}$$

Then

$$\mathbb{E}_{s \sim \bar{s}} [V^\pi(s)] - V^\pi(\bar{s}) \leq \Delta_Q \tag{8}$$

Proof of Lemma C.2.

$$\mathbb{E}_{s \sim \bar{s}} [V^\pi(s)] - V^\pi(\bar{s}) \tag{9}$$

$$= \int_s Pr(s|\bar{s}) \int_{\bar{a}} \Pr(\bar{a}|s) Q^\pi(s, \bar{a}) d\bar{a} ds - \int_{\bar{a}} \Pr(\bar{a}|\bar{s}) Q^\pi(\bar{s}, \bar{a}) d\bar{a} \tag{10}$$

$$= \int_s Pr(s|\bar{s}) \int_{\bar{a}} \Pr(\bar{a}|s) Q^\pi(s, \bar{a}) d\bar{a} ds - \int_{\bar{a}} \int_s \Pr(s|\bar{s}) \Pr(\bar{a}|s) Q^\pi(\bar{s}, \bar{a}) d\bar{a} ds \tag{11}$$

$$= \int_s \int_{\bar{a}} Pr(s|\bar{s}) \Pr(\bar{a}|s) \left(Q^\pi(s, \bar{a}) - Q^\pi(\bar{s}, \bar{a})\right) d\bar{a} ds \tag{12}$$

$$= \mathbb{E}_{s \sim \bar{s}} \left[\mathbb{E}_{\bar{a} \sim s} \left[Q^\pi(s, \bar{a}) - Q^\pi(\bar{s}, \bar{a})\right]\right] \tag{13}$$

$$\leq \mathbb{E}_{s \sim \bar{s}} \left[\max_{\bar{a}} \left(Q^\pi(s, \bar{a}) - Q^\pi(\bar{s}, \bar{a})\right)\right] \tag{14}$$

$$\implies \mathbb{E}_{s \sim \bar{s}} [V^\pi(s)] - V^\pi(\bar{s}) \leq \Delta_Q \tag{15}$$

■

Now, using the Bellman Equation for both $Q^\pi(s,\bar{a})$ and $Q^\pi(\bar{s},\bar{a})$, we set up a recurrence relation in order to prove our bound.

Proof of Recursive Relationship.

$$Q^\pi(s,\bar{a}) - Q^\pi(\bar{s},\bar{a}) \tag{16}$$

$$= R(s,\bar{a}) + \gamma^\tau \int_{s'} T^{s,\bar{a},s'} V^\pi(s')ds' - R(\bar{s},\bar{a}) - \gamma^\tau \sum_{\bar{s}'} T^{\bar{s},\bar{a},\bar{s}'} V^\pi(\bar{s}') \tag{17}$$

$$= \underbrace{R(s,\bar{a}) - R(\bar{s},\bar{a})}_{\leq \epsilon_R} + \gamma^\tau \int_{s'} T^{s,\bar{a},s'} V^\pi(s')ds' - \gamma^\tau \sum_{\bar{s}'} T^{\bar{s},\bar{a},\bar{s}'} V^\pi(\bar{s}') \tag{18}$$

$$\leq \epsilon_R + \gamma^\tau \int_{s'} T^{s,\bar{a},s'} V^\pi(s')ds' - \gamma^\tau \sum_{\bar{s}'} T^{\bar{s},\bar{a},\bar{s}'} \underbrace{V^\pi(\bar{s}')}_{\geq \mathbb{E}_{s'\sim\bar{s}'}[V^\pi(s') - \Delta_Q] \text{ (Lemma C.2)}} \tag{19}$$

$$\leq \epsilon_R + \gamma^\tau \int_{s'} T^{s,\bar{a},s'} V^\pi(s')ds' - \gamma^\tau \sum_{\bar{s}'} T^{\bar{s},\bar{a},\bar{s}'} \int_{s'} \Pr(s'|\bar{s}') \left(V^\pi(s') - \Delta_Q\right) ds' \tag{20}$$

$$= \epsilon_R + \gamma^\tau \int_{s'} T^{s,\bar{a},s'} V^\pi(s')ds' - \gamma^\tau \int_{s'} \underbrace{\sum_{\bar{s}'} T^{\bar{s},\bar{a},\bar{s}'} \Pr(s'|\bar{s}')}_{=T^{\bar{s},\bar{a},s'}} \left(V^\pi(s') - \Delta_Q\right) ds' \tag{21}$$

$$= \epsilon_R + \gamma^\tau \int_{s'} T^{s,\bar{a},s'} V^\pi(s')ds' - \gamma^\tau \int_{s'} T^{\bar{s},\bar{a},s'} \left(V^\pi(s') - \Delta_Q\right) ds' \tag{22}$$

$$= \epsilon_R + \gamma^\tau \Delta_Q + \gamma^\tau \int_{s'} \left(T^{s,o,s'} - T^{\bar{s},\bar{a},s'}\right) V^\pi(s')ds' \tag{23}$$

$$\leq \epsilon_R + \gamma^\tau \Delta_Q + \gamma^\tau \underbrace{\int_{s'} \left|T^{s,\bar{a},s'} - T^{\bar{s},\bar{a},s'}\right|}_{\leq \epsilon_T} \underbrace{|V^\pi(s')|}_{\leq \text{VMAX}} ds' \tag{24}$$

$$\leq \epsilon_R + \gamma^\tau \Delta_Q + \gamma^\tau \epsilon_T \text{VMAX} \tag{25}$$

$$\leq \epsilon_R + \gamma \Delta_Q + \gamma \epsilon_T \text{VMAX} \tag{26}$$

■

Since this holds for all $\bar{s}, o$, this implies that

$$\Delta_Q \leq \epsilon_R + \gamma \epsilon_T \text{VMAX} + \gamma \Delta_Q \tag{27}$$

$$\implies \Delta_Q \leq \frac{\epsilon_R + \gamma \epsilon_T \text{VMAX}}{1-\gamma} \tag{28}$$

Thus completes our proof.

D Proof for Theorem 4.1

To first define the abstraction transition function, we observe that transitions in the abstract MDP modify the factors contained in the relevant outcome's mask, causing the values of factors in the mask to be changed to the factor values representing its effect. All other factor values remain unchanged.

The corresponding abstract transition function is:

$$T(\bar{s}'|\bar{s},\bar{a}) = \begin{cases} p_i & \text{if } \bar{s}'[f_j] = e_j(\omega_i), \forall f_j \in f(\omega_i) \text{ and} \\ & \bar{s}'[f_k] = \bar{s}[f_k], \forall f_k \notin f(\omega_i) \text{ and} \\ & \bar{s}'_g = I'_{Oi} \\ 0 & \text{otherwise,} \end{cases} \tag{29}$$

where the outcome ω_i with mask m_i and initiation vector I'_{Oi} occurs with probability p_i (possibly 1) when executing action $\bar{a}$ from $\bar{s}$.

Theorem D.1. *Given an abstract factored option $\bar{a}$ and an abstract state $\bar{s}$ with grounding $\mathcal{G}_p(\bar{s})$, then $T(\bar{s}'|\bar{s},\bar{a})$ computed by Equation 29 accurately represents the distribution over states in which the agent may find itself after executing o from a state drawn from $\mathcal{G}_p(\bar{s})$.*

Proof of Theorem D.1. Given that abstract state $\bar{s}$ has grounding:

$$\mathcal{G}_p(\bar{s}) = P(s \mid I_O) = \Pi_{\omega_i \in \omega(\bar{s})} \left[\int \cdots \int_{\bar{e}(\omega_i, \bar{s})} P_i(s[m_i]) d \cdots d_{\bar{e}(\omega_i, \bar{s})} \right], \tag{30}$$

we seek to prove that the abstract transition function grounds to the distribution over states the agent may find itself after executing action $\bar{a}$ from a state drawn from $\mathcal{G}_p(\bar{s})$. By combining multiple factored outcomes and factored subgoal options in Definitions 3 and 4 with Definition 2, we arrive at:

$$T(s' \mid s \sim \bar{s}, o, I'_O) = \sum_{\omega_i} p_i \left[P_i(s'[m_i] \mid \bar{s}, \bar{a}, I'_{Oi}) \times P(s[\overline{m}_i] \mid I_O) \right],$$

where there are i outcomes. The abstract transition function has a similar form, via Equation 29: $T(\bar{s}'|\bar{s},\bar{a}) = \sum_i p_i \bar{s}'_i$. It therefore suffices to show that:

$$\mathcal{G}_p(\bar{s}'_i) = P_i(s'[m_i] \mid I'_{Oi}) \times P(s[\overline{m}_i] \mid I_O), \tag{31}$$

for each outcome ω_i. We begin by applying the grounding definition (Definition 8) to state $\bar{s}'_i$, which gives:

$$\mathcal{G}_p(\bar{s}'_i) = \Pi_{\omega_j \in \omega(\bar{s}'_i)} \left[\int \cdots \int_{\bar{e}(\omega_j, \bar{s}'_i)} P_j(s[m_j] \mid I_{Oj}) d \cdots d_{\bar{e}(\omega_j, \bar{s}'_i)} \right],$$

where $\omega(\bar{s}'_i)$ is the set of outcomes with factor values present in $\bar{s}'_i$, and $\bar{e}(\omega_j, \bar{s}'_i)$ is the set of factors that are in ω_j's mask but have factor values not in $\bar{s}'_i$. One outcome present in $\bar{s}'_i$ is ω_i itself, which we know by Equation 29 has every relevant factor value set in $\bar{s}'_i$:

$$\bar{s}'_i[f_j] = e_j(\omega_i), \forall f_j \in f(\omega_i),$$

and hence $\bar{e}(\omega_i, \bar{s}'_i)$ is empty, so:

$$\mathcal{G}_p(\bar{s}'_i) = P_i(s'[m_i] \mid I_{Oi}) \times \Pi_{\omega_j \in \omega(\bar{s}') \setminus \omega_i} \left[\int \cdots \int_{\bar{e}(\omega_j, \bar{s}')} P_j(s[m_j] \mid I_{Oj}) d \cdots d_{\bar{e}(\omega_j, \bar{s}')} \right]. \tag{32}$$

Comparing equations 31 and 32 we see that the left components of the product are the same; it remains to show that the right portions are equal. Specifically, we must show that:

$$P(s[\overline{m}_i] \mid I_O) = \Pi_{\omega_j \in \omega(\bar{s}') \setminus \omega_i} \left[\int \cdots \int_{\bar{e}(\omega_j, \bar{s}')} P_j(s[m_j] \mid I_{Oj}) d \cdots d_{\bar{e}(\omega_j, \bar{s}')} \right].$$

Since neither term is a function of any variables in m_i, we can freely introduce additional integrals over m_i. We must therefore prove:

$$P(s[\overline{m}_i] \mid I_O) = \int \cdots \int_{(m_i)} \Pi_{\omega_j \in \omega(\bar{s}') \setminus \omega_i} \left[\int \cdots \int_{\bar{e}(\omega_j, \bar{s}')} P_j(s[m_j] \mid I_{Oj}) d \cdots d_{\bar{e}(\omega_j, \bar{s}')} \right] d_{(m_i)}.$$

Next, we note that $\omega(\bar{s}') \setminus \omega_i \subseteq \omega(\bar{s})$, i.e., that every outcome in $\omega(\bar{s}') \setminus \omega_i$ is also in $\omega(\bar{s})$. Additionally, outcomes in $\omega(\bar{s})$ but not in $\omega(\bar{s}') \setminus \omega_i$ can only be functions of m_i, and so can be included in the product on the right hand side because they will subsequently be integrated out. We must therefore now prove that:

$$P(s[\overline{m}_i] \mid I_O) = \int \cdots \int_{(m_i)} \Pi_{\omega_j \in \omega(\bar{s})} \left[\int \cdots \int_{\bar{e}(\omega_j, \bar{s}')} P_j(s[m_j] \mid I_{Oj}) d \cdots d_{\bar{e}(\omega_j, \bar{s}')} \right] d_{(m_i)}.$$

Finally, we can replace $\bar{e}(\omega_j, \bar{s}')$ above with $\bar{e}(\omega_j, \bar{s})$ because these terms can only differ via factors in m_i, which are subsequently integrated out. Hence it remains to prove that:

$$\begin{aligned} P(s[\overline{m}_i] \mid I_O) &= \int \cdots \int_{(m_i)} \Pi_{\omega_j \in \omega(\bar{s})} \left[\int \cdots \int_{\bar{e}(\omega_j, \bar{s})} P_j(s[m_j] \mid I_{Oj}) d \cdots d_{\bar{e}(\omega_j, \bar{s}')} \right] d_{(m_i)} \\ &= \int \cdots \int_{(m_i)} \mathcal{G}_P(\bar{s}) \, d_{(m_i)}, \end{aligned}$$

which is its definition. ■

E Planning using An Abstract MDP

An abstract MDP $\overline{M}$ is sufficient for planning, after modification for a given *plan query*:

Definition 9. A *plan query* on the ground MDP is a tuple $Q_\tau = (B_\tau, G_\tau, R_\tau)$, where $B_\tau(s)$ is a start state distribution, $G_\tau \subseteq S$ is a set of terminal goal states, and $R_\tau : G_\tau \to \mathbb{R}$ is the reward for entering states in G_τ.

$\overline{M}$ must be modified as follows: first, execution should cease when the agent enters a state in G_τ, with terminating reward given by R_τ. Any abstract state $\overline{s}$ that overlaps G_τ must therefore be refined into a terminating abstract state contained in G_τ, and possibly a state representing $\overline{s}$'s remaining support. This process is depicted in Figure 8. The resulting goal states are grouped into goal set $\overline{G}$.

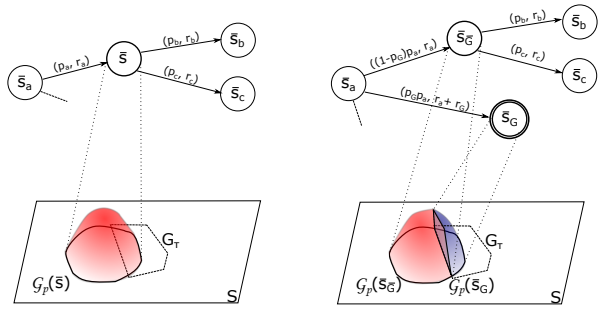


Figure 8: (a) Abstract state $\overline{s}$ with grounding distribution $\mathcal{G}_p(\overline{s})$ that overlaps grounded goal set $G_\tau \subseteq S$. (b) $\overline{s}$ must be refined into a goal state $\bar{s}_G$ that grounds to the portion of $\mathcal{G}_p(\overline{s})$ overlapping G_τ (with probability p_G), and $\bar{s}_{\overline{G}}$, which grounds to the remainder of $\mathcal{G}_p(\bar{s})$ (probability $1 - p_G$). $\bar{s}_G$ is terminating with expected goal reward r_G, while $\bar{s}_{\overline{G}}$ has the same reward and outgoing transitions as $\bar{s}$. Incoming transitions to $\bar{s}$ are distributed according to p_G.

Second, the agent must collect into start state set B all states with non-empty grounding overlap with B_τ. These need not be refined because the abstraction is approximately value preserving. Any planning algorithm can then be used to compute an optimal policy leading from any state in B to any in G.

F Limitations

In this work, we derive an algorithm to build an abstract MDP that can be iteratively refined to reduce the transition and reward error to an arbitrary value. Our implementation uses GMMs for such refinements. While partitioning an abstract state essentially decrease its size, and therefore makes it more likely to improve the model, it is possible to hit the local minima with the iterative clustering, and fail to refine further. On the other hand, our method is agnostic to the choice of the clustering approach used, and thus, we believe that this should not pose a serious limitation.

Our experiments in *Montezuma's Revenge* uses annotated RAM variables, While it is possible to train an MSA network on the RGB observations, the learned representations are not necessarily factored, which makes it hard to show how the algorithm operates. While this is not directly a limitation of the method—as our focus is on how to learn factored abstract representations from ground variables from given factors—it can be regarded as the limitation of the paper.

Currently, Algorithm 1 works in a batched fashion on a set of environment interactions. Ideally, it should iteratively collect data from states that are not visited frequently to expand its frontier. As we are interested in constructing the abstract MDP given enough data, we have used an expert policy in 10% of the transitions in *Montezuma's Revenge*, and deviate from the expert policy otherwise.

G Network Details

```
MSAFlat(
  (encoder): Sequential(
    (0): Linear(in_features=1568, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=N_LATENT, bias=True)
  )
  (inverse_fc): Sequential(
    (0): Linear(in_features=4, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=5, bias=True)
  )
  (density_fc): Sequential(
    (0): Linear(in_features=4, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=1, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=2, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=1568, bias=True)
  )
  (mi): Sequential(
    (0): Linear(in_features=4, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=1, bias=True)
  )
)
```

We used a two-layered multi-layer perceptron (MLP) with 64 hidden units and ReLU activations for the MSA encoder. The output dimensionality of the MSA encoder is set to 8 for Visual Maze domain, and to 2 for MNIST gridwalk domain. `inverse_fc` and `density_ec` modules compute the inverse model and the density ratio losses, respectively. `mi` is trained seperately (with a seperate 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.

```
QNetwork(
  (features_extractor): Linear(in_features=1568, out_features=64, bias=True)
  (q_net): Sequential(
    (0): Linear(in_features=4704, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=4, bias=True)
  )
)
```

We used the default structure of the DQN model that is provided in stable-baselines3[4] [51] with the only difference in the output dimensionality that depends on the number of actions in the domain. Also, we used the same feature extractor to process the observation and the desired goal, which considerably increased the performance, as opposed to plain concatenation of them. Note that the encoder of MSA and the Q network of the DQN has the same number of layers and hidden units.

H 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). As we are using the goal-conditioned DQN, we experimented with hindsight experience replay [8] and found that while it helped in MNIST gridwalk domain, it made the results worse for the Visual Maze domain. For MSA training, we only did an informal search of frequently used hyperparameter values (Table 1).

Table 1: MSA hyperparameters

Parameter	**Value**
Batch size	32
Validation split	0.1
Optimizer	Adam
Learning rate	0.001
β_1	0.9
β_2	0.999
ϵ	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 ϵ_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 ϵ_R	None
Minimum $\Delta\epsilon_R$ for refinement	None
n cluster trials	10
Min. samples for refinement	10

The main hyperparameters of Algorithm 1 (Table 2) are minimum transition and reward errors, ϵ_T, ϵ_R, and their relative improvements after a refinement, $\Delta\epsilon_T$, $\Delta\epsilon_R$. Using the transition error sufficed in our domains. ϵ_T and ϵ_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, ϵ 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.

I 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. 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 ϵ_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

[4] `https://stable-baselines3.readthedocs.io/en/master/`

Table 3: DQN hyperparameters

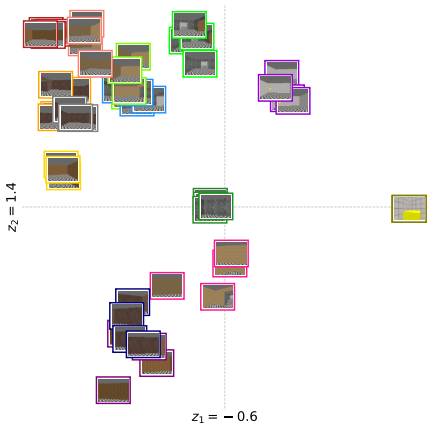
Parameter	**Value**
Batch size	32
Learning start step	100
Optimizer	Adam
Learning rate	0.0001
β_1	0.9
β_2	0.999
ϵ	10^{-8}
Soft update coeff. (τ)	1.0
Discount factor (γ)	0.99
Train frequency (every k step)	1
Gradient steps (every k rollout)	1
Replay Buffer	HER [8]
Number of sampled goals	4
Target update steps	10000
Exploration fraction	0.1
Exploration initial ϵ	1.0
Exploration final ϵ	0.05
Maximum gradient norm	10.0

approximations and with special data structures like k-d trees to reduce the time spent on finding nearest neighbors.

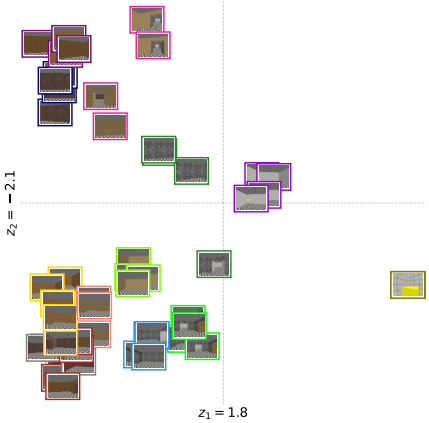
We used a GPU cluster to train multiple MSA networks in batch and build the abstract MDP locally on a laptop with Apple M1 Pro chip and 16 GB of unified memory. DQN models are trained locally on this same laptop and another with NVIDIA RTX 4070 GPU and Intel Core Ultra 9 Processor 185H. It is difficult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

J MSA Embedding Visualizations

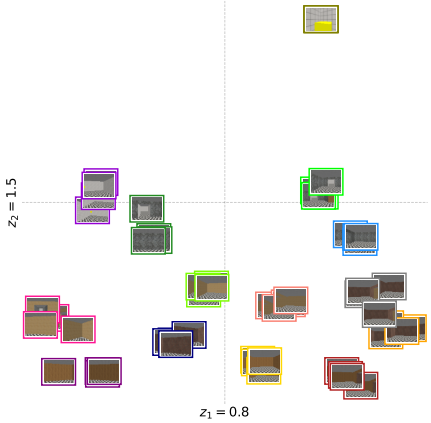
We visualized the learned embedding spaces in different domains for varying samples sizes in Figures 9, 10, 11, 12, and 13.



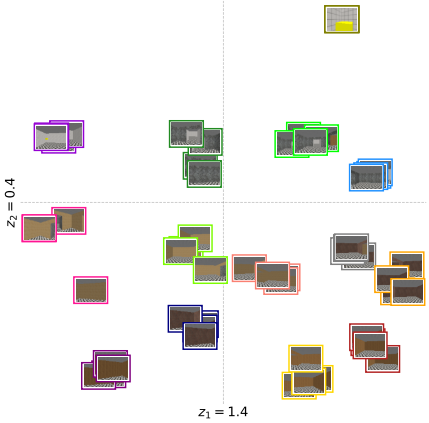
(a) 1000 samples, Seed 0



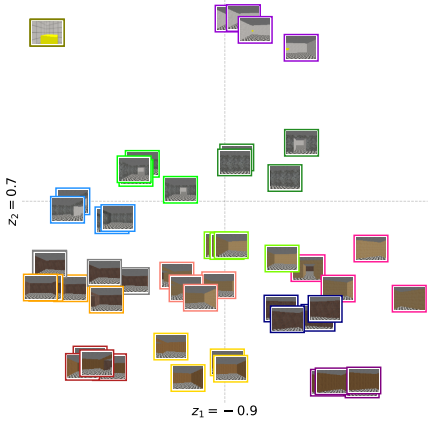
(b) 1000 samples, Seed 1



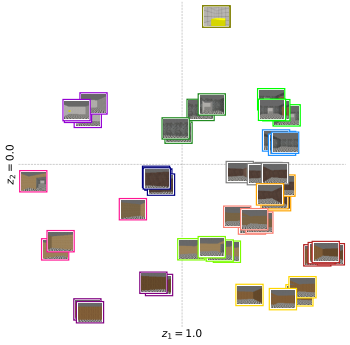
(c) 5000 samples, Seed 0



(d) 5000 samples, Seed 1



(e) 10000 samples, Seed 0



(f) 10000 samples, Seed 1

Figure 9: 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.

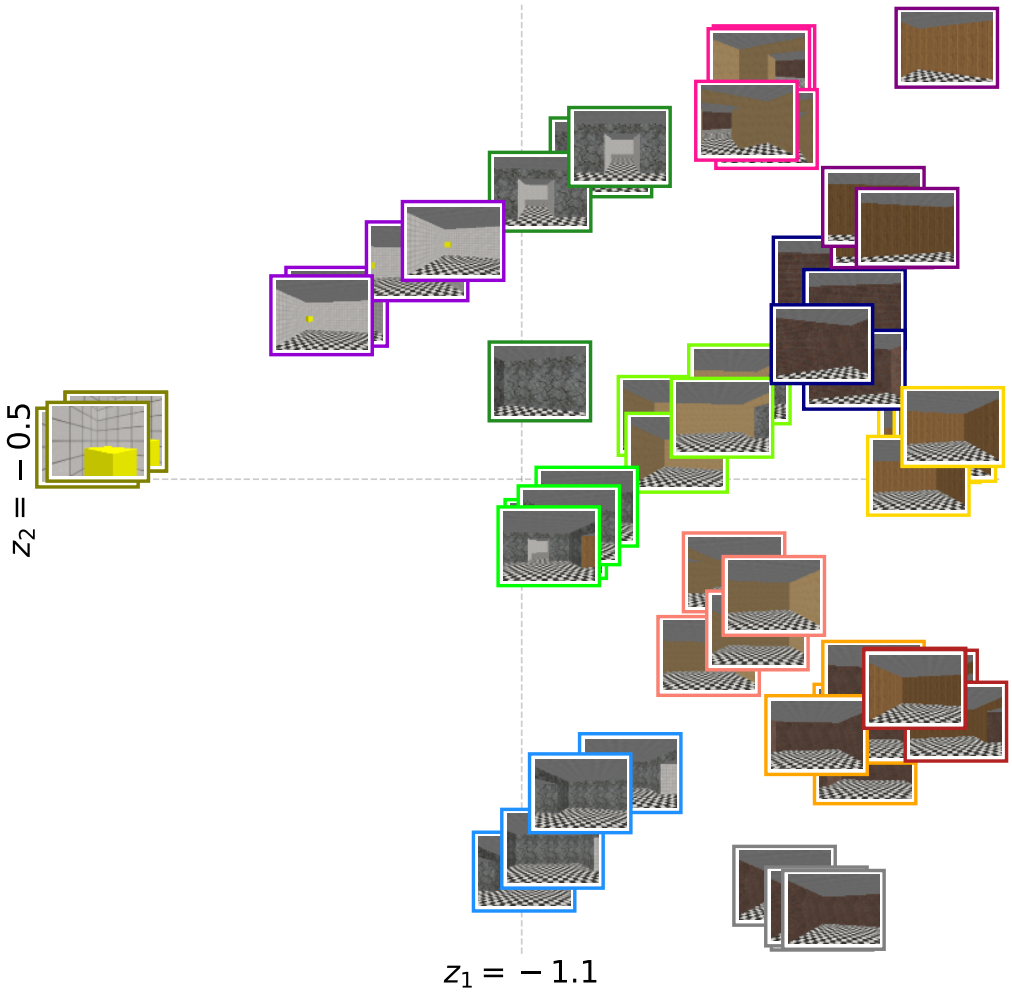


Figure 10: Learned MSA embedding space in the Visual Maze domain at 160×120 pixels.

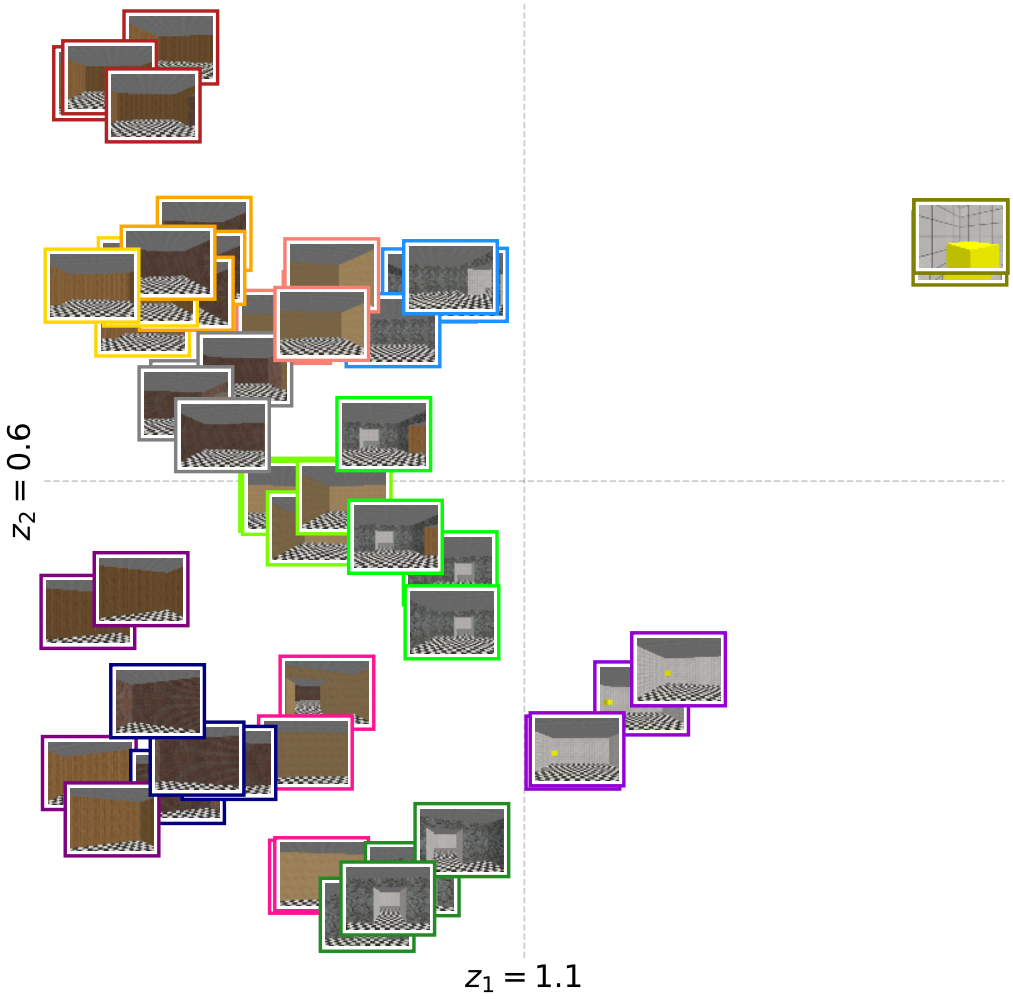


Figure 11: Learned MSA embedding space in the Visual Maze domain at 240×180 pixels.

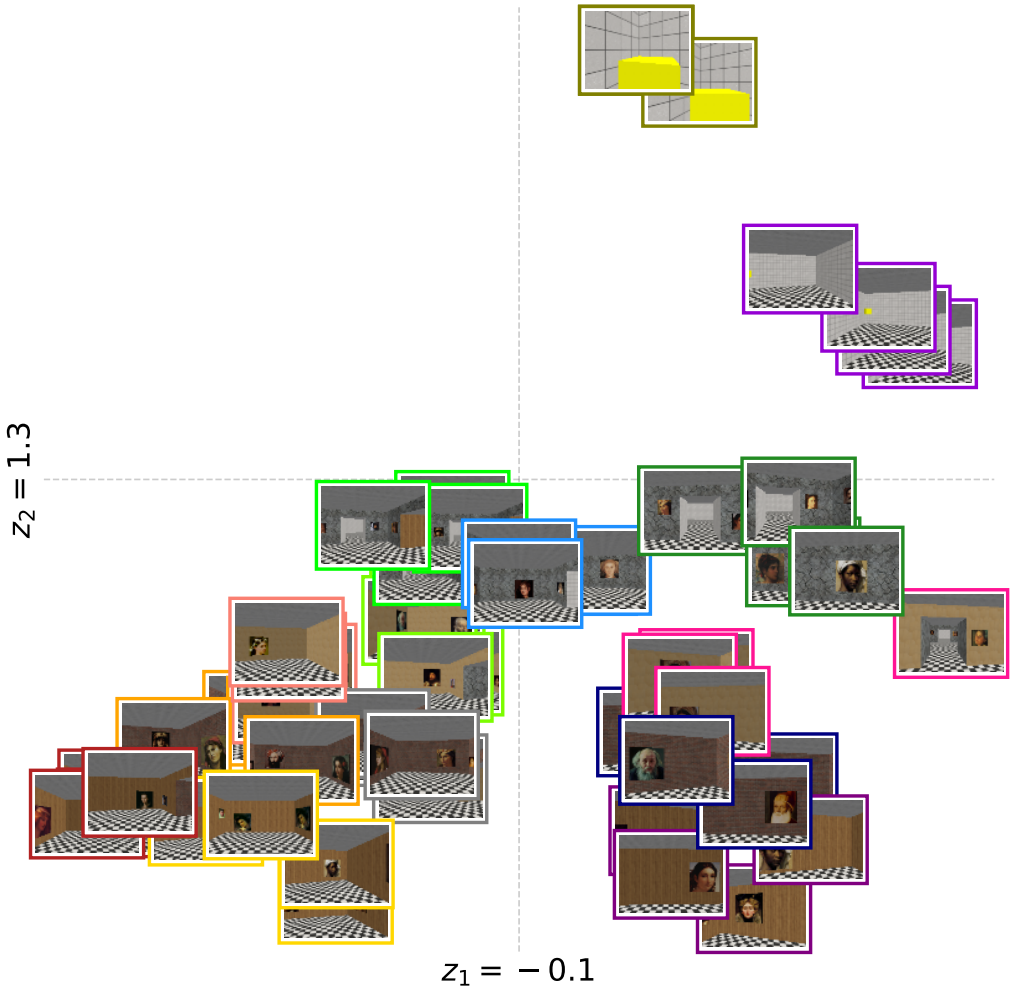
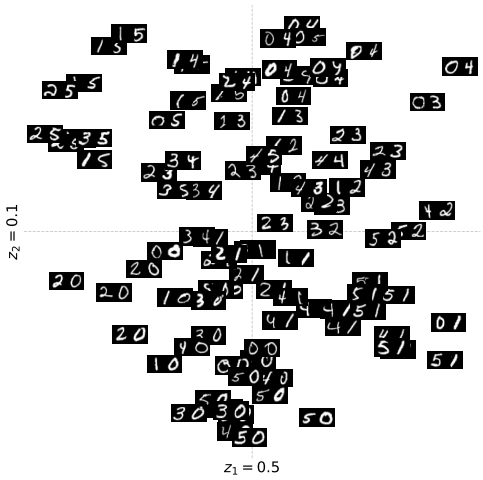
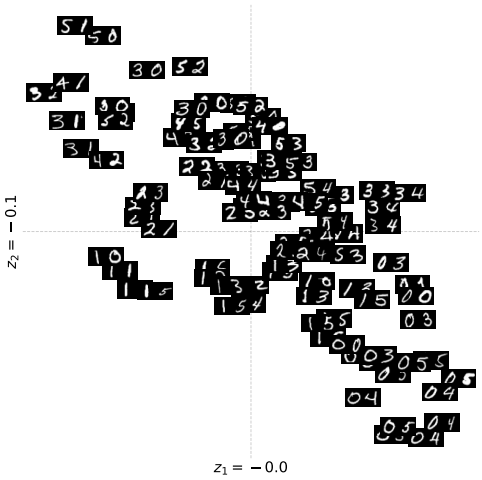


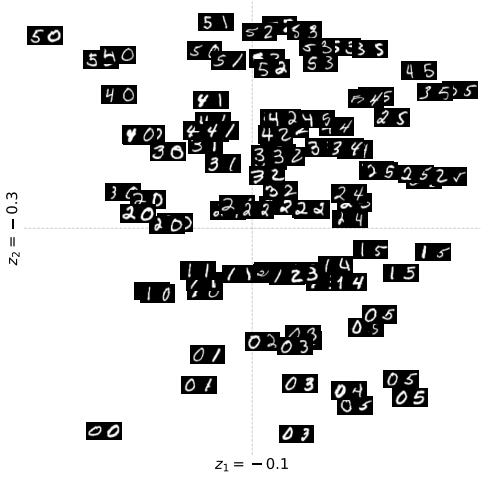
Figure 12: Learned MSA embedding space in the Visual Maze domain at 240×180 pixels with random portraits on the wall.



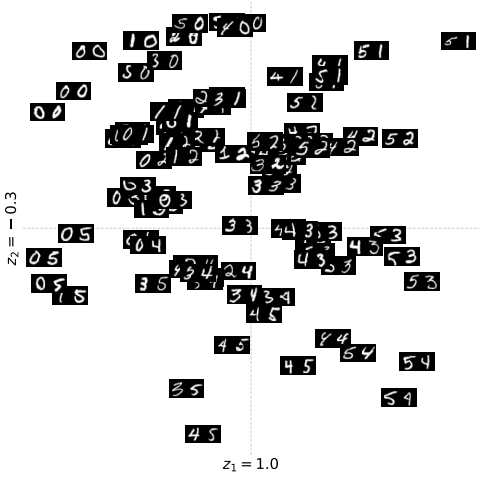
(a) 1000 samples, Seed 0



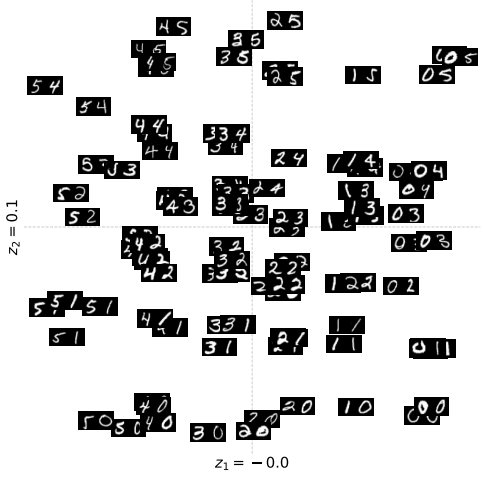
(b) 1000 samples, Seed 1



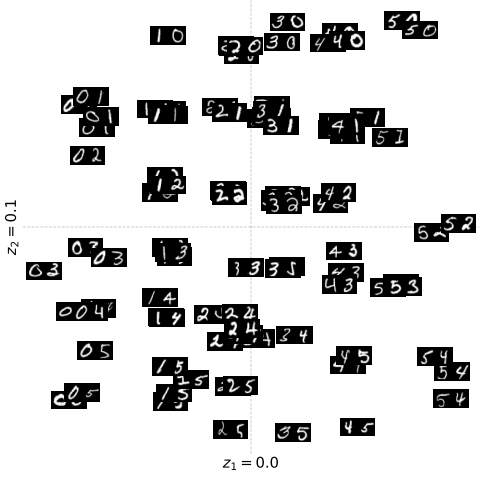
(c) 5000 samples, Seed 0



(d) 5000 samples, Seed 1



(e) 10000 samples, Seed 0



(f) 10000 samples, Seed 1

Figure 13: Learned MSA embedding spaces in the MNIST grid domain for different numbers of samples. 2-dimensional positions are the MSA encodings of input images.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We show why the Bellman equation suggests that abstract states should be distributions over ground states, provide an algorithm to build an abstract MDP with neural and symbolic components, and apply it on different domains.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We put limitations in the appendix due to the space constraints.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: Proofs for theoretical results are provided in the appendix.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Some of the details are provided in the main text, and others are reported in the appendix. We also provided pseudocodes for the important parts of the algorithm.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The source code is provided in the supplementary material with sufficient explanation to reproduce the results.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Hyperparameter details are provided in the supplementary material together with the source code.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We provide two standard errors of sample means which correspond a 96% confidence interval.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report the compute resources that are used throughout the experiments in the supplementary file.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This paper does not contain any experiments involving human subjects, and the domains we use in the experiments are common reinforcement learning environments that does not have any of the data-related concerns mentioned in the link.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: While there might be many potential societal consequences of our work, none which we feel must be specifically highlighted here as the proposed work is at a preliminary stage.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We reference the domains and datasets that are used in the manuscript.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Important parts of the source code are explained in a README file.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.