ABSTRACT
While reinforcement learning has achieved considerable successes in recent years, state-of-the-art models are often still limited by the size of state and action spaces. Model-free reinforcement learning approaches use some form of state representations and the latest work has explored embedding techniques for actions, both with the aim of achieving better generalization and applicability. However, these approaches consider only states or actions, ignoring the interaction between them when generating embedded representations. In this work, we establish the theoretical foundations for the validity of training a reinforcement learning agent using embedded states and actions. We then propose a new approach for jointly learning embeddings for states and actions that combines aspects of model-free and model-based reinforcement learning, which can be applied in both discrete and continuous domains. Specifically, we use a model of the environment to obtain embeddings for states and actions and present a generic architecture that leverages these to learn a policy. In this way, the embedded representations obtained via our approach enable better generalization over both states and actions by capturing similarities in the embedding spaces. Evaluations of our approach on several gaming, robotic control, and recommender systems show it significantly outperforms state-of-the-art models in both discrete/continuous domains with large state/action spaces, thus confirming its efficacy.

CSCS CONCEPTS
• Computing methodologies → Sequential decision making.

KEYWORDS
reinforcement learning, embeddings, representation learning

1 INTRODUCTION
Reinforcement learning (RL) has been successfully applied to a range of tasks, including challenging gaming scenarios [25]. However, the application of RL in many real-world domains is often hindered by the large number of possible states and actions these settings present. For instance, resource management in computing clusters [8, 22], portfolio management [16], and recommender systems [19, 21] all suffer from extremely large state/action spaces, thus challenging to be tackled by RL.

In this work, we investigate efficient training of reinforcement learning agents in the presence of large state-action spaces, aiming to improve the applicability of RL to real-world domains. Previous work attempting to address this challenge has explored the idea of learning representations (embeddings) for states or actions, either by adding additional layers to the RL agent’s network architecture or by using separately trained embedding models. Specifically, for state embeddings, using machine learning to obtain meaningful features from raw state representations is a common practice in RL, e.g. through the use of convolutional neural networks for image input [24]. Previous works such as Ha and Schmidhuber [13] have explored the use of environment models, termed world models, to learn state representations in a supervised fashion, and several pieces of literature explore state aggregation using bisimulation metrics [3]. While for action embeddings, the most recent works by Tennenholtz and Mannor [35] and Chandak et al. [4] propose methods for learning embeddings for discrete actions that can be directly used by an RL agent and improve generalization over actions. However, these works consider the state representation and action representation as isolated tasks, which ignore the underlying relationships between them. In this regard, we take a different approach and propose to jointly learn embeddings for states and actions, aiming for better generalization over both states and actions in their respective embedding spaces.

More importantly, even with state or action embedding as in some existing works, there is generally a lack of theoretical guarantee that ensures the learned policy in the embedding space can also achieve optimality in the original problem domain. To this end, we establish the theoretical foundations showing how the policy learned purely in the state and action embedding space is linked to the optimal policy in the original problem domain. We then propose an architecture consisting of two models: a model of the environment that is used to generate state and action representations and a model-free RL agent that learns a policy using the embedded states and actions. One key benefit of this approach is that state and action representations can be learned in a supervised manner, which greatly improves sampling efficiency and potentially enables their use for transfer learning. In sum, our key contributions are:
We consider an agent interacting with its environment over discrete time steps, where the environment is modelled as a discrete-time Markov decision process (MDP), defined by a tuple \((S, A, T, R, \gamma)\). \(S\) and \(A\) denote the state space and action space, respectively. In this work, we consider both discrete and continuous state and action spaces. The transition function from one state to another, given an action, is \(T: S \times A \mapsto S\), which may be deterministic or stochastic. The agent receives a reward at each time step defined by \(R: S \times A \mapsto \mathbb{R}, \gamma \in [0, 1]\) denotes the reward discounting factor. The state, action, and reward at time \(t \in \{0, 1, 2, \ldots \}\) are denoted by the random variables \(S_t, A_t, R_t\). The initial state of the environment comes from an initial state distribution \(d_0\). Therefore, the agent follows a policy \(\pi\), defined as a conditional distribution over actions given states, i.e., \(\pi(a|s) = P(A_t = a|S_t = s)\). The goal of the reinforcement learning agent is to find an optimal policy \(\pi^*\) that maximizes the expected sum of discounted accumulated future rewards for a given environment, i.e., \(\pi^* \in \arg \max_{\pi} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t|\pi]\). For any policy, we also define the state value function \(\mathbb{V}^\pi(s) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k}|\pi, S_t = s]\) and the state-action value function \(Q^\pi(s, a) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k}|\pi, S_t = s, A_t = a]\).

## 2 BACKGROUND

We consider an agent interacting with its environment over discrete time steps, where the environment is modelled as a discrete-time Markov decision process (MDP), defined by a tuple \((S, A, T, R, \gamma)\). \(S\) and \(A\) denote the state space and action space, respectively. In this work, we consider both discrete and continuous state and action spaces. The transition function from one state to another, given an action, is \(T: S \times A \mapsto S\), which may be deterministic or stochastic. The agent receives a reward at each time step defined by \(R: S \times A \mapsto \mathbb{R}, \gamma \in [0, 1]\) denotes the reward discounting factor. The state, action, and reward at time \(t \in \{0, 1, 2, \ldots \}\) are denoted by the random variables \(S_t, A_t, R_t\). The initial state of the environment comes from an initial state distribution \(d_0\). Therefore, the agent follows a policy \(\pi\), defined as a conditional distribution over actions given states, i.e., \(\pi(a|s) = P(A_t = a|S_t = s)\). The goal of the reinforcement learning agent is to find an optimal policy \(\pi^*\) that maximizes the expected sum of discounted accumulated future rewards for a given environment, i.e., \(\pi^* \in \arg \max_{\pi} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t|\pi]\). For any policy, we also define the state value function \(\mathbb{V}^\pi(s) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k}|\pi, S_t = s]\) and the state-action value function \(Q^\pi(s, a) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k}|\pi, S_t = s, A_t = a]\).

## 3 RELATED WORK

For the application of state embeddings in reinforcement learning, there are two dominant strands of research, namely world models and state aggregation using bisimulation metrics. World models approach trains an environment model in a supervised fashion from experience collected in the environment, which is then used to generate compressed state representations [12] or to train an agent using the learned world model [13, 30]. Further applications of world models, e.g., for Atari 2000 domains, show that abstract state representations learned via world models can substantially improve sample efficiency [15, 17]. Recent work by Ermolov and Sebe [7], Tao et al. [34] demonstrate that state embeddings learned via world models can also be used to guide exploration during training, either based on the error of transition predictions [7] or based on the distance of state embeddings in latent space [34]. Similar to this idea, Munk et al. [26] pre-train an environment model and use it to provide state embeddings for an RL agent. Furthermore, de Bruin et al. [5] investigate additional learning objectives to learn state representations, and Francois-Lavet et al. [10] propose the use of an environment model to generate abstract state representations; their learned state representations are then used by a Q-learning agent. By using a learned model of the environment to generate abstract states, these approaches capture structure in the state space and reduce the dimensionality of its representation. In contrast to world models, we consider both states and actions and train an RL agent on the original environment using its embedded representation rather than using a surrogate world model. Bisimulation, on the other hand, is a method for aggregating states that are “behaviorally equivalent” [20] and can improve convergence speed by grouping similar states into abstract states. Several works on bisimulation-based state aggregation, e.g., Ferns et al. [9], Givan et al. [11], present different metrics to measure state similarity. Furthermore, Zhang et al. [39] and Castro [3] propose deep learning methods for generating bisimulation-based state aggregations that scale beyond the tabular methods proposed in several earlier works [3]. While there are parallels between bisimulation and our approach, we do not propose the aggregation of states. Instead, our embedding technique projects states into a continuous state embedding space, similar to Zhang et al. [39], where their behavioral similarity is captured by their proximity in embedding space. Furthermore, our method embeds both states and actions and does not employ an explicit similarity metric such as a bisimulation metric, but instead learns the relationships among different states and actions via an environment model. State representations are also used in RL-based NLP tasks, such as Narasimhan et al. [27], who jointly train an LSTM-based state representation module and a DQN agent, and Ammanabrolu and Riedl [2], who propose the use of a knowledge graph based state representation.

In addition to state representations, previous work has explored the use of additional models to learn meaningful action representations. In particular, Van Hasselt and Wiering [37] investigate the use of a continuous actor in a policy gradient based approach to solve discrete action MDPs, where the policy is learned in continuous space and actions are discretized before execution in the environment. Dulac-Arnold et al. [6] propose a similar methodology, where a policy is learned in continuous action embedding space and then mapped to discrete actions in the original problem domain. Both Van Hasselt and Wiering [37] and Dulac-Arnold et al. [6] only consider actions and assume that embeddings are known a priori. Tennenholtz and Mannor [35] propose a methodology called Act2Vec, where they introduce an embedding model similar to the Skip-Gram model [23] that is trained using data from expert demonstrations and then combined with a DQN agent. One significant drawback of this approach is that information on the semantics of actions has to be injected via expert demonstration and is not learned automatically. In contrast, Chandak et al. [4], Whitney et al. [38] propose methods that enable the self-supervised learning of state and action representations from experience collected by the RL agent. Chandak et al. [4] use an embedding model that resembles
With these embedding spaces, we further define an internal policy $\pi_i$, together with the internal policy $\pi_i$ form the overall policy $\pi$, which then allow us to remove the original state and action from the expression of the policy, leaving us with $\pi_i$ is related to $\pi$, after which we prove the existence of $\pi_0$ that achieves optimality in the original problem domain. For this goal, two further assumptions are required on the nature of the state embedding function $\phi$ and the action mapping function $f$.

**Assumption 1.** Given an action embedding $E_t$, $A_t$ is deterministic and defined by a function $f : E \mapsto A$, i.e., there exists action a such that $P(A_t = a | E_t = e) = 1$.

**Assumption 2.** Given a state $S_t$, $X_t$ is deterministic and defined by a function $\phi : S \mapsto X$, i.e., there exists state embedding $x$ such that $P(X_t = x | S_t = s) = 1$.

We validate Assumption 1, which defines $f$ as a many-to-one mapping, empirically for all experiments conducted in Section 5, and find that no two actions share exactly the same embedded representation, i.e., Assumption 1 holds in practice. Note that we also assume the Markov property for our environment, which is a standard assumption for reinforcement learning problems. With slight abuse of notation, we denote the inverse mapping from an action $a \in A$ to its corresponding points in the embedding space (one-to-many mapping) by $f^{-1}(a) = \{e \in E : f(e) = a\}$.

**Lemma 1.** Under Assumptions 1 and 2, for policy $\pi$ in the original problem domain, there exists $\pi_i$ such that

$$v^\pi(s) = \sum_{a \in A} \int_{f^{-1}(a)} \pi_i(e|x = \phi(s)) Q^\pi(s,a) \, de .$$

**Proof Sketch.** Based on the Bellman equation for the value function $v^\pi$ in the original domain, we introduce the embedded state $x = \phi(s)$ using Assumption 2. By the law of total probability and Assumption 1, we then introduce the embedded action $e$. From the Markovian property and the definition of our model, we can derive Claims 1 - 5 on (conditional) independence in Appendix A.1, which then allow us to remove the original state and action from the expression of the policy, leaving us with $\pi_i$. See Appendix A.2 for the complete proof.

By Assumptions 1 and 2, Lemma 1 allows us to express the overall policy $\pi_0$ in terms of the internal policy $\pi_i$ as

$$\pi_0(a | s) = \int_{f^{-1}(a)} \pi_i(e | x = \phi(s)) \, de ,$$

bridging the gap between the original problem domain and the policy in embedding spaces $X$ and $E$. Under $\pi^*$ in the original domain,
we define \( \varphi := \psi^T \) and \( Q^* := Q^T \) for the ease of discussions. Then using Lemma 1, we now prove the existence of an overall policy \( \pi_o \) that is optimal.

**Theorem 1.** Under Assumptions 1 and 2, there exists an overall policy \( \pi_o \) that is optimal, such that \( \varphi_{\pi_o} = \varphi^* \).

**Proof.** Under finite state and action sets, bounded rewards, and \( \gamma \in (0, 1) \), at least one optimal policy \( \pi^* \) exists. From Lemma 1, we then have

\[
\pi^*(s) = \sum_{a \in A} \int_{(e) = f^{-1}(a)} \pi_i(e|\phi(s))Q^*(s, a) \, de .
\]  

Thus, \( \exists \phi, f, \) and \( \pi_i \), representing an overall policy \( \pi_o \), which is optimal, i.e., \( \varphi_{\pi_o} = \varphi^* \). \( \Box \)

Theorem 1 suggests that in order to get the optimal policy \( \pi^* \) in the original domain, we can focus on the optimization of the overall policy \( \pi_o \), which is discussed in the next section.

### 4.3 Architecture Embedding and Training

In this section, we first present the implementation and the training of the state-action embedding model (illustrated in Figure 1b). Based on this method and according to Theorem 1, we then propose a strategy to train the overall policy \( \pi_o \), where functions \( \phi \) and \( f \) are iteratively updated.

#### 4.3.1 Joint Training of the State-Action Embedding

The difference between the true transition probabilities \( P(S_{t+1}|S_t, A_t) \) and the estimated probabilities \( \hat{P}(S_{t+1}|S_t, A_t) \) can be measured using the Kullback-Leibler (KL) divergence, where the expectation is over the true distribution \( P(S_{t+1}|S_t, A_t) \), i.e.,

\[
D_KL(P||\hat{P}) = -\mathbb{E}_{S_{t+1} \sim P(S_{t+1}|S_t, A_t)} \ln \left( \frac{\hat{P}(S_{t+1}|S_t, A_t)}{P(S_{t+1}|S_t, A_t)} \right) .
\]  

From tuples \( (S_{t+1}, S_t, A_t) \), we can compute a sample estimate of this using Equation (4). In Equation (4), the denominator does not depend on \( \hat{\phi}, \hat{g}, \) or \( \hat{T} \). Therefore, we define the loss function for the embedding model as

\[
\mathcal{L}(\hat{\phi}, \hat{g}, \hat{T}) = -\mathbb{E}_{S_{t+1} \sim P(S_{t+1}|S_t, A_t)} \left[ \ln(\hat{P}(S_{t+1}|S_t, A_t)) \right] .
\]  

Note that the estimator \( \hat{f} \) is not directly included in the embedding model. Instead, we find \( \hat{f} \) by minimizing the error between the original action and the action reconstructed from the embedding. We can define this as

\[
\mathcal{L}(\hat{f}) = -\ln(\hat{f}(A_t|E_t)) .
\]

Then, all components of our embedding model can be learned by minimizing the loss functions \( \mathcal{L}(\hat{\phi}, \hat{g}, \hat{T}) \) and \( \mathcal{L}(\hat{f}) \). Note that the embedding model is trained in two steps, firstly by minimizing the loss in Equation (5) to update \( \hat{\phi}, \hat{g}, \) and \( \hat{T} \) and secondly by minimizing the loss in Equation (6) to update \( \hat{f} \). Here, we give the target of the embedding model for the case of discrete state and action spaces. However, the model is equally applicable to continuous domains. In this case, the loss functions \( \mathcal{L}(\hat{\phi}, \hat{g}, \hat{T}) \) and \( \mathcal{L}(\hat{f}) \) are replaced by loss functions suitable for continuous domains. For continuous domains in Section 5, we adopt a mean squared error loss instead of the losses given in Equations (5-6), but other loss functions may also be applicable. For experiments, we parameterize all components of the embedding model illustrated in Figure 1b as neural networks. More details can be found in Appendix A.4.

#### 4.3.2 State-Action Embedding and Policy Learning

Theorem 1 only shows that optimizing \( \pi_o \) can help us achieve optimality in the original domain. Then a natural question is whether we can optimize \( \pi_i \) by directly optimizing the internal policy \( \pi_i \) using the state/action embeddings in Section 4.3.1. To answer this question, we first mathematically derive that updating \( \pi_i \) is equivalent to updating \( \pi_o \). We then proceed to present an iterative algorithm for learning the state-action embeddings and the internal policy.
Suppose \( \pi_t \) is parameterized by \( \theta \). Then with the objective of optimizing \( \pi_o \) by only updating \( \pi_t \), we define the performance function of \( \pi_o \) as:

\[
J_o(\phi, \theta, f) = \sum_{s \in S} d_o(s) \sum_{a \in A} \pi_o(a|s) Q^o(s, a).
\]  

(7)

Let the state-action-value function for the internal policy be \( Q^i(x, e) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k}|\pi_t(X_t = x, E_t = e)] \). We can then define the performance function of the internal policy as:

\[
J_i(\theta) = \int_{x \in X} \int_{e \in \mathcal{E}} d_i(x) \pi_i(e|x) Q^i(x, e) \, de \, dx.
\]  

(8)

The parameters of the overall policy \( \pi_o \) can then be learned by updating its parameters \( \theta \) in the direction of \( \frac{\partial J_o(\phi, \theta, f)}{\partial \theta} \), while the parameters of the internal policy can be learned by updating \( \theta \) in the direction of \( \frac{\partial J_i(\theta)}{\partial \theta} \). With the following assumption, we then have Lemma 2.

**Assumption 3.** The state embedding function \( \phi(s) \) maps each state \( s \) to a unique state embedding \( x = \phi(s) \), i.e., \( \forall s_i \neq s_j, P(\phi(s_i) = \phi(s_j)) = 0 \).

Note that Assumption 3, defining \( \phi \) as a *one-to-one* mapping, is not theoretically guaranteed by the definition of our embedding model. Nevertheless, we test this empirically in Section 5 and find that no two states share the same embedded representation in any of our experiments, thus justifying Assumption 3 in practical scenarios considered in this work.

**Lemma 2.** Under Assumptions 1–3, for all deterministic functions \( f \) and \( \phi \), which map each point \( e \in \mathcal{E} \) and \( s \in S \) to an action \( a \in A \) and to an embedded state \( x \in X \), and the internal policy \( \pi_i \) parameterized by \( \theta \), the gradient of the internal policy’s performance function \( \frac{\partial J_i(\theta)}{\partial \theta} \) equals the gradient of the overall policy’s performance function \( \frac{\partial J_o(\phi, \theta, f)}{\partial \theta} \), i.e.,

\[
\frac{\partial J_i(\theta)}{\partial \theta} = \frac{\partial J_o(\phi, \theta, f)}{\partial \theta}.
\]

\(^1\)Equation (7) is for the overall policy with discrete states and actions. Note that our approach is also applicable in continuous domains. \( \sum_{s \in S} \) and \( \sum_{a \in A} \) would then be replaced by \( \int_s \) and \( \int_a \) respectively.

Algorithm 1 Joint Training of State-Action Embeddings and the Internal Policy

\[
\text{Initialize the state and action embeddings (optional pre-training)}
\]

\[
\text{for Epoch = 0, 1, 2, \ldots} \quad \text{do}
\]

\[
\text{for } t = 1, 2, 3, \ldots \quad \text{do}
\]

\[
\text{Generate state embedding } X_t = \phi(S_t), \text{ Sample action embedding } E_t \sim \pi_i(S_t), \text{ Map embedded action to } A_t = f(E_t), \text{ Execute } A_t \text{ in the environment to observe } S_t, R_t, \text{ Update the } \pi_i \text{ and the critic using some policy gradient algorithms}, \text{ Update } \phi, g, T, \text{ and } f \text{ by minimizing the losses in Equations (5) and (6)}
\]

\end{align*}

**5 EMPIRICAL EVALUATION**

We evaluate our proposed architecture, denoted by JSAE (Jointly-trained State Action Embedding), on game-based applications, robotic
control, and a real-world recommender system, covering different combinations of discrete and continuous state and action spaces. Our methodology is evaluated using Vanilla Policy Gradient (VPG) [1], Proximal Policy Optimization (PPO) [31], and Soft Actor Critic (SAC) [14] algorithms. We benchmark the performance against these algorithms without embeddings and with action embeddings generated by the method called policy gradients with Representations for Actions (RA) proposed by Chandak et al. [4]. To isolate the effect of dimensionality reduction, we additionally benchmark against the aforementioned algorithms with embeddings generated from two auto-encoders (AE) – one for states and one for actions. The used auto-encoders consist of a simple two-layer feed-forward network, where the weights of the first layer after training are used as embeddings. We pre-train the embedding models on randomly collected samples for all experiments and enable continuous updates for the Ant-v2 and recommender system environments.

5.1 Proof-of-Concept: Gridworld and Slotmachine

Gridworld: It is similar to that used by Chandak et al. [4]. States are given as a continuous coordinate, while actions are defined via \( n \) actuators, equally spaced around the agent, which move the agent in the direction they are pointing towards. Then each combination of actuators forms an action, resulting in \( 2^n \) unique actions. We run two sets of experiments in this environment: (i) we use the continuous state directly and (ii) we discretize the coordinate in a \( 40 \times 40 \) grid. The agent receives small step and collision penalties and a large positive reward for reaching the goal state. Figure 2 illustrates the learned embeddings for this environment. The obtained embeddings for states and actions illustrate that our approach is indeed able to obtain meaningful representations. Actions are embedded according to the displacement in the Cartesian coordinates they represent (Figure 2d). Figure 2b shows that states are embedded according to the coordinate they represent and interestingly, the embeddings capture the \( L \)-shaped obstacle in the original problem domain.

Slotmachine: It consists of four reels with 6 values per reel, each of which can be individually spun by the agent for a fraction of a full turn. Then, each unique combination of these fractions constitutes an action and the agent receives rewards for aligning reels. The discrete state is given by the numbers on the front of the reels.

Both scenarios yield large discrete action spaces, rendering them well-suited for evaluating our approach. The results for the gridworld and slot machine environments are shown in Figure 3a, 3b, and 3c. Our approach (JSAE) outperforms all three benchmarks on the discretized gridworld and the slot machine environments and is comparable for the continuous-state gridworld. Such observation suggests that our approach is particularly well suited for discrete domains. Intuitively, this is because the relationship between different states or actions in discrete domains is usually not apparent, e.g., in one-hot encoding. In continuous domains, however, structure in the state and action space is already captured to some extent, rendering it harder to uncover additional structure by embeddings; nevertheless, when a continuous problem becomes more complicated, our embedding method again shows effectiveness in capturing useful information (see Section 5.2). Notably, the auto-encoder generated embeddings lead to worse performance on both gridworld domains and only slightly improve convergence speed on the slot machine environment. As discrete state and action spaces do not necessarily contain an inherent structure, the auto-encoder can only reduce the dimensionality without capturing the underlying structure of the problem domain, and potentially embed behaviorally different states/actions close to each other. Such initial findings suggest that it is not the dimensionality reduction, but our approach’s ability to capture the environment structure in the embedding space that improves the performance.

5.2 Robotic Control

We use the half- cheetah-v2 and the Ant-v2 environments from the robotic control simulator Mujoco [36] and limit the episode length to 250 and 1000 steps respectively. Here, the agent observes continuous states (describing the current position of the cheetah or ant respectively) and then decides on the force to apply to each of the joints of the robot (i.e., continuous actions). Rewards are calculated based on the distance covered in an episode. On these domains, the auto-encoder baseline performs comparable to the baselines without embeddings (Figure 3d, 3e). While the auto-encoder is now able to preserve much of the inherent structure of the state-action space, as we have continuous state and action spaces, it is not able to use the interaction between states and actions to structure the embedding space. From the results presented in Figure 3d, we observe that our method (JSAE) does not outperform the benchmarks on the half-cheetah-v2 environment. On the Ant-v2 environment, however, our approach outperforms the benchmarks. Similar to the experiments in a continuous gridworld, it is less likely that embedding uncovers additional structure on top of the structure inherent to the continuous state and action, explaining the comparable performance with and without embeddings on the half-cheetah-v2 domain. Nevertheless, in the more challenging Ant-v2 environment, the state and action representation is more complex and the agent might benefit from additional structure uncovered via our approach. By contrasting our approach with the auto-encoder baseline, we can again confirm that the improved performance does not stem from a mere dimensionality reduction in the state and action representations.

5.3 Recommender System

In addition, we also test our methodology on a real-world application of a recommender system. We use data on user behavior from an e-commerce store collected in 2019 [18]. The environment is constructed as an \( n \)-gram based model, following Shani et al. [32]. Based on the \( n \) previously purchased items, a user’s purchasing likelihood is computed for each item in the store, leading to a stochastic transition function. Recommending an item then scales the likelihood for that item by a pre-defined factor. States are represented as a concatenation of the last \( n \) purchased items and each item forms an action. In this environment we have 835 items (actions) and approx. 700,000 states. The results obtained under various RL methods are reported in Figure 3f. Note that we do not run an SAC benchmark, as this algorithm is designed for continuous action spaces only. We find that our approach leads to significant improvements in both the convergence speed and final performance compared to the benchmarks. This result confirms that our method is particularly useful in the presence of large discrete state and action spaces. Interestingly,
the PPO and VPG benchmarks outperform the benchmarks using the RA and AE methodologies. We conjecture that the action embeddings generated in RA and AE on this environment in fact obfuscate the effect an action has, and thus limit the agent’s performance.

5.4 Application in Transfer Learning
In addition to improving the convergence speed and the accumulated reward, we test whether the policies learned in one environment can be transferred to a similar one. We consider two environments (old and new) that differ by only the state space or the action space. To leverage the previous experience, if the two environments have the same state (or action) space, the state (or action) embedding component in the new environment is initialized with the weights learned from the old environment; we then train our model from scratch, in terms of the convergence speed. Thus, it shows that the previously learned policy serves as a critical initialization for the new environment and sheds light on the potential of our approach in transfer learning; further evaluations are left for future work.

6 CONCLUSION
In this paper, we presented a new architecture for jointly training state/action embeddings and combined this with common reinforcement learning algorithms. Our theoretical results confirm the validity of the proposed approach, i.e., the existence of an optimal policy in the embedding space. We empirically evaluated our method on several environments, where it outperforms state-of-the-art RL approaches in complex large-scale problems. Our approach is easily extensible as it can be combined with most existing RL algorithms and there are no special restrictions on embedding model parameterizations.

A APPENDIX
A.1 Claims
In addition to Assumptions 1 and 2, we derive three claims on conditional independence from the definition of our embedding model and the Markovian property of the environment, listed below.

Claim 1. Action \( A_t \) is conditionally independent of \( S_t \) given a state embedding \( X_t \):

\[
P(A_t = a, S_t = s | X_t = x) = P(A_t = a | X_t = x)P(S_t = s | X_t = x).
\]

Claim 2. Action \( A_t \) is conditionally independent of \( S_t \) given state and action embeddings \( X_t \) and \( E_t \):

\[
P(A_t = a, S_t = s | X_t = x, E_t = e) = P(A_t = a | X_t = x, E_t = e)P(S_t = s | X_t = x, E_t = e).
\]
Claim 3. Next state $S_{t+1}$ is conditionally independent of $E_t$ and $X_t$ given $A_t$ and $S_t$:

$$P(S_{t+1} = s' \mid E_t = e, X_t = x, A_t = a, S_t = s) = P(S_{t+1} = s' \mid A_t = a, S_t = s) P(E_t = e \mid X_t = x, A_t = a, S_t = s).$$

Using Claims 1 - 3, we can derive two further auxiliary claims that will be used in the proof of Lemma 1.

Claim 4. The probability of next state $S_{t+1}$ is independent of $E_t$ and $X_t$, given state $S_t$ and action $A_t$, i.e.,

$$P(S_{t+1} = s' \mid E_t = e, X_t = x, A_t = a, S_t = s) = P(S_{t+1} = s' \mid A_t = a, S_t = s).$$

Proof. From Claim 3, we have

$$P(s' \mid e, x, a, s) = P(s' \mid a, s) P(e, x \mid a, s)$$

$$P(s' \mid e, x, a, s) = P(s' \mid a, s) P(e \mid x, a, s)$$

$$P(s' \mid e, x, a, s) = P(s' \mid a, s).$$

□

Claim 5. The probability of action $A_t$ is independent of $S_t$ and $X_t$, given action embedding $E_t$, i.e.,

$$P(A_t = a \mid S_t = s, X_t = x, E_t = e) = P(A_t = s \mid E_t = e).$$

Proof. From Claim 2, we have

$$P(a \mid x, e) = P(a \mid x, e) P(x \mid a, e)$$

$$P(a \mid x, e) = P(a \mid x, e) .$$

Since action $a$ only depends on the action embedding $e$ in our model, this becomes

$$P(a \mid x, e) = P(a \mid e) .$$

□

A.2 Proof of Lemma 1

Lemma 1. Under Assumptions 1 and 2, for policy $\pi$ in the original problem domain, there exists $\pi_i$ such that

$$v^\pi(s) = \sum_{a \in A} \int_{e=f^{-1}(a)} \pi_i(e \mid x = \phi(s)) Q^\pi(s, a) \, de$$

Proof. The Bellman equation for a MDP is given by

$$v^\pi(s) = \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} P(s' \mid s, a) G(s', a)$$

where $G$ denotes the return, i.e. $[R(s, a) + \gamma v^\pi(s')]$, which is a function of $s$, $a$, and $s'$. By re-arranging terms we get

$$v^\pi(s) = \sum_{a \in A} \sum_{s' \in S} \pi(a \mid s) P(s' \mid s, a) G(s', a)$$

$$= \sum_{a \in A} \sum_{s' \in S} \pi(a \mid s) \frac{P(s', a \mid s)}{P(s, a)} G(s', a)$$

$$= \sum_{a \in A} \sum_{s' \in S} \pi(a \mid s) \frac{P(s', a \mid s)}{P(s, a) P(s, s)} G(s', s)$$

$$= \sum_{a \in A} \sum_{s' \in S} P(a \mid s) P(s' \mid s) G(s', s).$$

Since $s$ can be deterministically mapped to $x$ via $x = \phi(s)$, by Assumption 2, we have

$$v^\pi(s) = \sum_{a \in A} \sum_{s' \in S} P(x, a | s) P(s' \mid s) G(s', s)$$

$$= \sum_{a \in A} \sum_{s' \in S} \frac{P(x, a | s') P(s' \mid s) G(s', s)}{P(x \mid s')}$$

$$= \sum_{a \in A} \sum_{s' \in S} \frac{P(x, a, s') P(s' \mid s) G(s', s)}{P(x, s)}$$

$$= \sum_{a \in A} \sum_{s' \in S} P(x \mid a, s') P(s' \mid s) G(s', s)$$

$$= \sum_{a \in A} \sum_{s' \in S} P(x \mid a, s') P(s' \mid s) G(s', s)$$

$$= \sum_{a \in A} \sum_{s' \in S} P(x \mid a, s') P(s' \mid s) G(s', s)$$

From Claim 1, we know that $P(a \mid s, x) = P(a \mid x)$. Therefore,

$$v^\pi(s) = \sum_{a \in A} \sum_{s' \in S} P(s' \mid a, x) P(a \mid x) G(s', s)$$

Since $P(x \mid s)$ is deterministic by Assumption 2 and evaluates to 1 for the representation of $\phi(s) = x$, we can rewrite the equation above to

$$v^\pi(s) = \sum_{a \in A} \sum_{s' \in S} P(s' \mid a, x) P(a \mid x) G(s', s).$$

We now proceed to establish the relationship with the action embedding. From above we have

$$v^\pi(s) = \sum_{a \in A} \sum_{s' \in S} P(a \mid x) P(s' \mid a, x, s) G(s', s)$$

$$= \sum_{a \in A} \sum_{s' \in S} P(a \mid x) P(s' \mid a, x, s) G(s', s).$$

From Claim 1, $a$ and $s$ are conditionally independent given $x$. Therefore, $P(a, x, s) = P(a \mid x) P(s \mid x) P(x)$, which allows us to rewrite the above equation as

$$v^\pi(s) = \sum_{a \in A} \sum_{s' \in S} P(a \mid x) P(s' \mid a, x, s) G(s', s)$$

$$= \sum_{a \in A} \sum_{s' \in S} P(a \mid x) P(s' \mid a, x, s) G(s', s).$$

By the law of total probability, we can now introduce the new variable $e$, which is the embedded action. Then

$$v^\pi(s) = \sum_{a \in A} \sum_{s' \in S} \int_{e} P(a \mid e, s' \mid x, s) P(s' \mid x, s) G(s', s) \, de$$

$$= \sum_{a \in A} \sum_{s' \in S} \int_{e} P(e \mid x) P(a \mid e, s' \mid x, s) P(s' \mid x, s) G(s', s) \, de.$$
Since \( e \) is uniquely determined by \( x \), we can drop \( s \) in \( P(e|x,s) \), and thus

\[
\nu^\pi(s) = \sum_{a \in A} \sum_{s' \in S} \int \nu^\pi(a, e, s', x, s) \frac{P(a, e, s', x, s)}{P(e, x, s)} G \, de \approx \sum_{a \in A} \sum_{s' \in S} \int \nu^\pi(a, e, s', x, s) G \, de \approx \sum_{a \in A} \sum_{s' \in S} \int \nu^\pi(a, e, s', x, s) P(a|e, s) G \, de .
\]

Using the previously derived Claims 4 and 5, the above equation can be simplified to

\[
\nu^\pi(s) = \sum_{a \in A} \sum_{s' \in S} \int P(e|\phi(s))P(s'|a,s)P(a|e) G \, de .
\]

Since the function \( f \), mapping \( e \) to \( a \), is deterministic by Assumption 1 and only evaluates to 1 for a particular \( a \) and 0 elsewhere, we can rewrite this further as

\[
\nu^\pi(s) = \sum_{a \in A} \sum_{s' \in S} \int P(e|\phi(s))P(s'|a,s)G \, de .
\]

Summarizing the terms, this becomes

\[
\nu^\pi(s) = \sum_{a \in A} \int f^{-1}(a) \pi_i(e|\phi(s))Q^\pi(a,s) \, de .
\]

### A.3 Proof of Lemma 2

**Lemma 2.** Under Assumptions 1–3, for all deterministic functions \( f \) and \( \phi \) which map each point \( e \in \mathcal{E} \) and \( s \in S \) to an action \( a \in A \) and to an embedded state \( x \in X \), and the internal policy \( \pi_i \) parameterized by \( \theta \), the gradient of the internal policy’s performance function \( \frac{\partial J_0(\phi, \theta, f)}{\partial \theta} \) equals the gradient of the overall policy’s performance function \( \frac{\partial J_0(\phi, \theta, f)}{\partial \theta} \), i.e.,

\[
\frac{\partial J_0(\phi, \theta, f)}{\partial \theta} = \frac{\partial J_\pi(\phi, \theta, f)}{\partial \theta} .
\]

**Proof.** Recall from Lemma 1 that the overall policy is defined using the internal policy

\[
\pi_\pi(a|s) = \int f^{-1}(a) \pi_i(e|\phi(s)) \, de .
\]

We can then define the performance function of the overall policy using the internal policy as

\[
J_\pi(\phi, \theta, f) = \sum_{s \in S} \sum_{a \in A} \int f^{-1}(a) \pi_i(e|\phi(s))Q^\pi(a,s) \, de .
\]

The gradient of this performance function w.r.t. \( \theta \) is

\[
\frac{\partial J_\pi(\phi, \theta, f)}{\partial \theta} = \sum_{s \in S} \sum_{a \in A} \int f^{-1}(a) \pi_i(e|\phi(s))Q^\pi(a,s) \, de .
\]

Now consider another policy \( \pi'_i \) as illustrated in Figure 5, i.e., a policy that takes the raw state input \( S_t \) and outputs \( E_t \) in embedding space. By Assumption 1, \( Q^\pi_\pi(s,a) = Q^\pi_\pi(i(s), e) \) as \( e \) is deterministically mapped to \( a \). However, \( Q^\pi_\pi(x,e) = \mathbb{E}_{x \sim f^{-1}(x)}[Q^\pi_\pi(s,a)] = \mathbb{E}_{x \sim f^{-1}(x)}[Q^\pi_\pi(s,a)] \), where \( f^{-1}(\cdot) \) is the inverse mapping from \( x \) to \( s \). Nevertheless, by Assumption 3, \( \mathbb{E}_{x \sim f^{-1}(x)}[Q^\pi_\pi(s,a)] \)
