A Appendix

A.0 Related Work Continued

Discussion continued from Section 7:

Batch Reinforcement Learning. In recent times, [16] and [13] propose datasets and experimental setups for studying offline RL problems. A large body of work examines solutions to the batch RL problem. Researchers have identified that *extrapolation error*, the phenomenon in which batch RL algorithms incorrectly estimate the value of states/actions not present in the training batch, is a major challenge, and have proposed methods to tackle it, *e.g.* BCQ [14], BEAR [19], IRIS from [25], and CQL [21] among many others. In contrast to these model-free methods, [4, 31, 30] learn a forward predictive model from the batch data and use it for model predictive control. These methods all approach the traditional batch RL problem, while we consider a different and harder setting in which the action labels are unavailable. Aforementioned advances in offline RL are complementary to our work. Offline value learning approaches (such as CQL and BCQ) can serve as a drop-in replacement for Q-learning in our pipeline and improve our results. In fact, our experiments with BCQ substantiate this.

State-only Learning. In line of work which studies learning form *goal-directed* state-only experience, researchers use imitation learning-based techniques [29, 40, 11, 18], learn policies that match the distribution of visited states [41, 42, 43], or use demonstrations to construct dense reward functions [35, 34, 36, 46, 9]. These methods make strong assumptions about the quality and goal-directed nature of the experience data, and suffer in performance when faced with low-quality or undirected experience.

Instead of goal-directed experience our work tackles the problem of learning from undirected experience. Past work in this area employs Q-learning to learn optimal behavior from sub-optimal data [8, 37, 10]. [8] and [37] use domain specific insights. [10] rely on being able to generate the next state and only demonstrate results in environments with low-dimensional states. Instead, our work maps transition tuples to discrete latent actions and can thus easily work with high-dimensional observations such as RGB images.

A.1 Proofs

Proof. **Proof for Lemma 3.2.** We will start by showing that for any policy $\pi_{\hat{M}}$ on $\hat{M}$, there exists a policy π_M on M such that $V_{\hat{M}}^{\pi_{\hat{M}}}(s) = V_M^{\pi_M}(s)$, for all s.
To do this we need to introduce the idea of *fundamental actions*, which are classes of actions which have the same state and reward transition distributions in a given state. If we have a fundamental action b, corresponding to some state and reward distributions, let $\alpha(b, s) \subseteq \mathbf{A}$ give the set of actions in $\mathbf{A}$ that have the matching state and reward transition distributions,

$$\underset{a\in\alpha(b,s)}{\forall} p(s', r|s, a) = p(s', r|s, \alpha(b, s)_1).$$

Similarly, let $\hat{\alpha}(b, s) \subseteq \hat{\mathbf{A}}$ give the set of actions in $\hat{\mathbf{A}}$ belonging to b in state s. In any given state, there are at most min($|\mathbf{A}|, |\hat{\mathbf{A}}|$) fundamental actions for M and the union of all actions belonging to all fundamental actions gives the set of actions that make up the original action space. For our MDP let's denote $B(s)$ as the set of fundamental actions in the state s for M, and $\hat{B}(s)$ as the set of fundamental actions in the state s for $\hat{M}$. Let $\beta(s, a)$, and $\hat{\beta}(s, a)$ be functions which return the set of actions which correspond to the same fundamental action containing as a in state s for M, and $\hat{M}$ respectively. This means,

$$\bigcup_{b\in B(s)} \alpha(b, s) = A, \underset{b\in B(s)}{\forall} \underset{b'\neq b}{\forall} \alpha(b, s) \cap \alpha(b', s) = \emptyset, \text{ and } \underset{a\in A,s}{\forall} \exists_{b\in B(s)} | a \in \alpha(b, s). \tag{1}$$

With this notation out of the way we can construct a policy $\pi_{\hat{M}}$ from π_M, which achieves the same value in $\hat{M}$ as π_M does in M. We do this by constructing $\pi_{\hat{M}}$ such that the probability distributions over each *fundamental action* is equivalent. We define this policy as

$$\pi_M(a|s) = \frac{1}{|\beta(s, a)|} \sum_{\hat{a}\in\hat{\beta}(s,a)} \pi_{\hat{M}}(\hat{a}|s).$$

Following [38] we can define the value function for a given policy as

$$V_M^{\pi_M}(s) = \mathbb{E}_{\pi_M}[G_t | S_t = s],$$

$$V_M^{\pi_M}(s) = \sum_{a\in\mathbf{A}} \pi_M(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V_M^{\pi_M}(s')\big].$$

From the properties in Eq. 1, we know a sum over fundamental actions will count each action exactly once, so we can write this sum as

$$V_M^{\pi_M}(s) = \sum_{b\in B(s)} \sum_{a\in\alpha(b,s)} \pi_M(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V_M^{\pi_M}(s')\big]. \tag{2}$$

substituting in the constructed policy we have

$$V_M^{\pi_M}(s) = \sum_{b\in B(s)} \sum_{a\in\alpha(b,s)} \left[\frac{1}{|\beta(s, a)|} \sum_{\hat{a}\in\hat{\beta}(s,a)} \pi_{\hat{M}}(\hat{a}|s)\right] \sum_{s',r} p(s', r|s, a)\big[r + \gamma V_M^{\pi_M}(s')\big].$$

since the third sum is over $\hat{\beta}(s, a)$ with $a \in \alpha(b, s)$, we can rewrite this sum as over $\hat{a} \in \hat{\alpha}(b, s)$, giving

$$V_M^{\pi_M}(s) = \sum_{b\in B(s)} \sum_{a\in\alpha(b,s)} \left[\frac{1}{|\beta(s, a)|} \sum_{\hat{a}\in\hat{\alpha}(b,s)} \pi_{\hat{M}}(\hat{a}|s)\right] \sum_{s',r} p(s', r|s, a)\big[r + \gamma V_M^{\pi_M}(s')\big].$$

For the final sum, from the definition of fundamental actions, we know $p(s', r|s, a) = p(s', r|s, \alpha(b, s)_1) = \hat{p}(s', r|s, \hat{\alpha}(b, s)_1)$, so we can rewrite that term as

$$\sum_{s',r} \hat{p}(s', r|s, \hat{\alpha}(b, s)_1)\big[r + \gamma V_M^{\pi_M}(s')\big].$$

Crucially, in the second sum (over $a \in \alpha(b, s)$), $\beta(s, a)$ is the same for all $a \in \alpha(b, s)$, so the term in brackets can be treated as a constant. Similarly, since $|B(s, a)| = |\alpha(b, s)|$ we can simplify the second sum leaving

$$V_M^{\pi_M}(s) = \sum_{b \in B(s)} \sum_{\hat{a} \in \hat{\alpha}(b,s)} \pi_{\hat{M}}(\hat{a}|s) \sum_{s',r} \hat{p}(s', r|s, \hat{\alpha}(b, s)_1)\big[r + \gamma V_M^{\pi_M}(s')\big].$$

This gives the definition of $V_{\hat{M}}^{\pi_{\hat{M}}}(s)$ using the same decomposition as equation Eq. 2, meaning

$$V_M^{\pi_M}(s) = V_{\hat{M}}^{\pi_{\hat{M}}}(s).$$

One can show the opposite direction, that a policy there exists a policy $\pi_{M'}$ for any policy π_M such $V_M^{\pi_M}(s) = V_{\hat{M}}^{\pi_{\hat{M}}}(s)$, with a symmetric construction.

□

Note that Lemma 3.2 holds true regardless of the stochasticity of the policy, as the constructed policy matches probability of the original policy for taking each fundamental action.

Proof for Theorem 3.1. Under the new MDP $\hat{M}$, Q-learning will learn the same optimal value function value function as learned under MDP M, *i.e.* $\forall_s V_{\hat{M}}^*(s) = V_M^*(s)$.

Proof. This follows from Lemma 3.2 by contradiction. Assume there is a state s' where $V_{\hat{M}}^*(s) \neq V_M^*(s)$. Two cases must be considered. In the first case, $V_{\hat{M}}^*(s) > V_M^*(s)$. We will notate the optimal policy in $\hat{M}$ as $\tilde{\pi}_{\hat{M}}^*$ (*i.e.* $V_{\hat{M}}^* = V_{\hat{M}}^{\tilde{\pi}_{\hat{M}}^*}$) and the optimal policy in M as π_M^*. We know there must exist a policy $\tilde{\pi}_M^*$ such that $V_M^{\tilde{\pi}_M^*}(s) = V_{\hat{M}}^{\tilde{\pi}_{\hat{M}}^*}(s)$ from Lemma 3.2. We arrive at a contradiction because $V_M^{\tilde{\pi}_M^*}(s) > V_M^{\pi_M^*}(s)$, so π_M^* could not have been the optimal policy on M. The other direction follows a symmetric argument. If $V_{\hat{M}}^{\tilde{\pi}_{\hat{M}}^*}(s) < V_M^{\pi_M^*}(s)$, we know there must be a policy $\pi_{\hat{M}}^*$ in $\hat{M}$ such that $V_M^{\pi_M^*}(s) = V_{\hat{M}}^{\pi_{\hat{M}}^*}(s)$. This implies $V_{\hat{M}}^{\pi_{\hat{M}}^*}(s) > V_{\hat{M}}^{\tilde{\pi}_{\hat{M}}^*}(s)$, meaning that $\tilde{\pi}_{\hat{M}}^*$ could not have been the optimal policy. Since Q-learning is known to converge to the optimal value function [45], Q-learning on $\hat{M}$ will converge to the original value function V_M^*. □

A.2 State Only Learning From Stochastic MDPs

Here we will briefly prove that problem of deducing the optimal value function of an MDP M, from some state-only experience dataset D, cannot be solved in general for non-deterministic MDPs. We notate $\mathbb{D}$ as the space of all *complete* state-only datasets, and $\mathbb{V}$ as the space of all value functions. By *complete* state-only dataset, we mean datasets in which all possible transition triples (s, s', r) appear. Note that exactly one complete dataset exists for each MDP. We denote D_M as the complete dataset corresponding to an MDM M.

In short, the idea is to construct a single state-only dataset which ambiguously could have been produced from two different MDPs which have different optimal value functions. Since no function deterministic function can model two outputs from the same input, no such function can exist.

Theorem A.1. *For any function $f : \mathbb{D} \rightarrow \mathbb{V}$ which maps from the set of of state-only datasets, $\mathbb{D}$, to a set of value functions. There exists some MDP M, with corresponding dataset D_M such that $f(D) \neq V_M^*$.*

$$\nexists f \mid \forall_M f(D_M) = V_M^*.$$

Proof. We proceed by contradiction, assume any $f : \mathbb{D} \rightarrow \mathbb{V}$ such that

$$\forall_M f(D_M) = V_M^*.$$

If we can construct M_1 and M_2 such that $V_{M_1}^* \neq V_2^*$, and D such that $D = D_{M_1} = D_{M_2}$, then f cannot produce the correct value function for $f(D)$ in all cases, from the definition of a function. We will now show a very simple construction which exhibits this property. Consider an MDP M_1, with 3 states, s_1, s_2, s_t, and 2 actions a_1, a_2. s_t is terminal, and gives reward 1, other states give reward 0. The transition dynamics of the two actions are given below.

Table: a_1 transitions for M_1

	s_1	s_2	s_t
s_1	0	1	0
s_2	0	0	1

Table: a_2 transitions for M_1

	s_1	s_2	s_t
s_1	0	0	1
s_2	1	0	0

For simplicity, assume M_1 has a discount factor of 0.9. This means $V^*(s_1) = V^*(s_2) = 1$.

We define M_2 as identical to M_1, only with stochastic transitions.

Table: a_1 transitions for M_2

	s_1	s_2	s_t
s_1	0	1	0
s_2	0	0	1

Table: a_2 transitions for M_2

	s_1	s_2	s_t
s_1		0.9	0.1
s_2	1	0	0

In this MDP, with a discount factor of 0.9, $V^*(s_2) = 1$, but $V^*(s_1) = 0.1 + 0.9 * 0.9 = 0.91$.

Since both M_1 and M_2 share the same states, actions, and possible transitions, $D_{M_1} = D_{M_2}$. So we have satisfied our condition that $D = D_{M_1} = D_{M_2}$, and $V_{M_1}^* \neq V^* M_2$. Thus, no f can model the value functions for both MDPs in general. $\square$

A.3 Grid World Behavior *vs.* Purity

Figure S7 plots the MSE in value function and the proportion of states with correct implied actions as a function of the noise in the action labels.

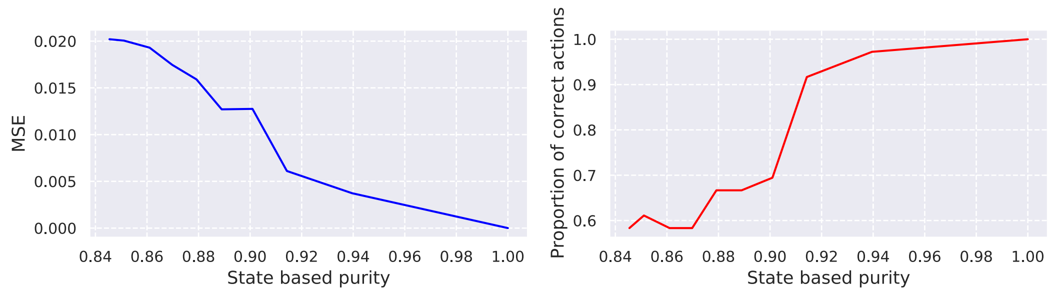

Figure S7: We plot correctness of value function estimate and behavior as a function of state-based purity of the intervening actions used for Q-learning. Left plot shows the mean squared error (lower is better) between the obtained value function and the optimal value function. Right plot shows fraction of states in which the learned value function induces the optimal action (higher is better). Both value function and behaviors become worse as state-based purity decreases.

A.4 Environment Details

2D Grid World. We use the environment and data from Section 3.2. We use the (x, y) coordinates as our state for this environment. We use a multi-layer perceptron for f_θ and L2 loss for l.

2D Continuous Control Navigation. We use the Maze2D data from D4RL [13]. Observation space here is (x, y, v_x, v_y) *i.e.* location and velocity of the agent, and action space is force along x and y directions. We use a frame skip of 3. We use a multi-layer perceptron for f_θ and L2 loss for l. In the embodiment transfer variant, we swap the agent for a 4-legged ant, also from D4RL. The ant embodiment is far more challenging to control, with an 8-d action space and 29-d observation space.

Atari Game. We work with Freeway. We generated our own data using the protocol described in [1]. We turned off sticky actions and store frames both at the default resolution (84×84) and full resolution (224×224). Other settings are same as is typical: stack last 4 frames to represent the observations, and use frame-skip of 4. We use a convolutional encoder-decoder model for f_θ and predict raw future observations. We use L2 distance in the pixel space as our loss function l.

3D Visual Navigation. We work with the branching environment from [8] in the AI Habitat Simulator [33], and use the provided dataset of first-person trajectories as our pre-recorded experience dataset. Following [8], we employ low-level controllers for deriving behaviors from the learned value functions. We use a convolutional encoder-decoder for f_θ, and L2 loss in pixel space as l.

Kitchen Manipulation. In this task a 9-DOF Franka arm can interact with several different objects physically simulated kitchen. The observations are 24-d, containing the end-effector position and state of various objects in the environment (microwave door angle, kettle position, etc.). The action space represents the 9-d joint velocity. In cross-embodiment experiments we replace the Franka arm with a hook. The observation space remains the same while the action space becomes 3-d position control.

For experiments in kitchen manipulation, we add an additional component to the reward computation to incentivize behavior to remain within the region of the state space covered by the demonstrations (where the learned value function is in-distribution). To do this, we build a density model over the end effector position using a 2 component Gaussian mixture model. If a state s has a less than 1% probability according to the GMM density model, then we assign $V(s) = 0$, for the update described in Section 5.2. We apply this shaping to both ours and D3G in Figure 4. We give sparse reward the same benefit of this shaping by giving reward -1 outside of the GMM distribution, and 0 inside. Sparse task reward remains the same.

For all environments, we throw out the provided action labels when learning our models.

A.5 Data Collection

2D Grid World. The setting for this experiment is a 6×6 grid with 8 actions, corresponding to moving in the 4 cardinal directions and 4 diagonal directions. The agent starts in the top left (0,0), and gets reward 0 everywhere, except for in the bottom right, (5,5). Reaching the bottom right gives the agent reward 1 and terminates the episode. In the starting square (0,0), the agent has probability 0.5 of moving right, and probability 0.5 of moving down. When the agent is on the top or bottom edge of the maze (row = 0 or row = 5) it moves right with probability 0.9, and takes a random action with probability 0.1. When it is on the left or right edge of the grid (column = 0 or column = 5), it moves down with probability 0.9 and takes a random action with probability 0.1. Otherwise, it is in the interior of the grid, where it takes an action away from the goal (randomly chosen from: up, left, or up-left) with probability 0.9, and one of the remaining actions otherwise (randomly chosen from: down, right, down-right, up-right, down-left). This policy is rolled out for 20,000 policies to generate the data for methods described in main paper Section 3.2 and Figure 1.

Atari Game. Data for the Freeway environment had to be additionally collected, as native high resolution resolution images of the episodes from [1] are not readily available. For this, we re-generated the data using the protocal described in [1], without sticky actions. Then, the high resolution data was collected by taking the action sequences from the re-generated dataset and executing them in an emulator, while storing the native resolution images. Because we re-generated the data without sticky actions, the high resolution episodes are perfect recreations of the low resolution episodes.

Visual Navigation. The data used for the visual navigation experiment was generated using the same protocol as [8]. Agents are tasked to reach one of two goals (G_{near} and G_{far}) in a visually realistic simulator [33]. Navigating to G_{near} is the optimal path as it is nearer to the agent's starting location.

The dataset contains 3 types of trajectories, T_1, T_2, and T_3, representing 50%, 49.5%, and 0.5% of the trajectories in the dataset respectively . T_1 takes a sub-optimal path to G_{near}, T_3 takes the optimal path to G_{near}, and T_2 navigates to G_{far}.

2D Continuous Control (Maze2D) We use the *Maze2D* dataset from D4RL as is.

Kitchen Manipulation. The data used for the Kitchen Manipulation environment comes from the *partial* version of the D4RL FrankaKitchen dataset. To facilitate cross-embodiment transfer we convert the state representation to contain end-effector location instead of joint angles.

A.6 Latent Action Quality

We first measure the effectiveness of our latent action mining process by judging the extent to which the induced latent actions are a refinement of the original actions. We measure this using the *state-conditioned purity* of the partition induced by the learned latent actions.

In a given state, for any latent action $\hat{a}$, there must be some ground truth action which most frequently mapped to $\hat{a}$. We define the purity of $\hat{a}$ as the proportion of the most frequent action among all actions mapped to $\hat{a}$. For example, in a given state s if a set of actions $[0, 0, 0, 1, 2]$, were mapped to latent action $\hat{a}$, then 0 is the most frequent action mapped to $\hat{a}$ and thus the purity of $\hat{a}$ would be 0.6. For a given state, the purity of an entire set of latent actions is the weighted mean of purity of individual latent actions. Overall purity is the average of all state wise purities weighted by how often the states appear in the dataset.

We extend this definition of state-conditioned purity to continuous or high-dimensional states by measuring the validation accuracy of a function g that is trained to map the high-dimensional state (or observation) s, and the associated latent action $\hat{a}$ to the actual ground truth action a. Training such a function induces an implicit partition of the state space. Learning to predict the ground truth action from the latent action $\hat{a}$ within this induced partition estimates the most frequent ground truth action, and accuracy measures its proportion, *i.e.* purity. This exact procedure reduces to the above definition for discrete state spaces, but also handles continuous and high-dimensional states well. For continuous action spaces, we measure the mean squared error in action prediction instead of classification accuracy.

Table S3 reports the purity (and MSE for continuous action environment) obtained by our proposed future prediction method. For reference, we also report the purity obtained when using *single action* that maps all samples into a single cluster, and 2 *clustering* methods that cluster either concatenation of the two observations *i.e.* $[o_{t+1}; o_t]$, or the difference between the two observations *i.e.* $[o_{t+1} - o_t]$. We use 8 latent actions for all environments except the FrankaKitchen environment for which we use 64 because of its richer underlying action space.

In general, our forward models are effective at generating a latent action space that is a state-conditioned refinement of the original action space. This is indicated by the improvement in state-conditioned purity values over using a single action or naive clustering. For the 2D Grid World and 2D Continuous Navigation, clustering in the correct space (state difference *vs.* state concatenation) works well as expected. But our future prediction model, which directly predicts s_{t+1} and doesn't use any domain specific choices, is able to outperform the corresponding clustering method. We also observe large improvements over all baselines for the challenging case of environments with high-dimensional state representations: Freeway and 3D Visual Navigation.

Table S3: We report the state-conditioned action purity (higher is better, MSE for continuous action case where lower is better), of latent actions for different approaches: single action, clustering concatenated observations, clustering difference in observations, and the proposed future prediction models from Section 4.1. We note the utility of the future prediction model for the challenging case of Freeway and 3D Visual Navigation environments. See Section A.6 for a full discussion.

Environment	**Observation Space**	**Action Space**	**Purity Metric**	**Single Action**	**Clustering** $[o_t, o_{t+1}]$	**Clustering** $[o_{t+1} - o_t]$	**Future Prediction**
2D Grid World	xy location	Discrete, 8	Purity (↑)	0.827	0.851	**1.000**	0.998
Freeway	210×160 image	Discrete, 3	Purity (↑)	0.753	0.778	0.773	**0.907**
3D Visual Navigation (Branching)	224×224 image	Discrete, 3	Purity (↑)	0.783	0.839	0.859	**0.928**
2D Continuous Control	xy loc. & vel	Continuous, 2	MSE (↓)	2.207	2.188	**0.325**	0.905
Kitchen Manipulation	24-d State	Continuous, 8	MSE (↓)	0.015	0.015	0.015	**0.014**

A.7 Analysis of Learned Latent Actions

We analyze the latent actions learned for the Freeway environment.

We visualize our future prediction model's predictions for the Freeway environment in Figure S8. In line with our expectations, the one action future prediction model and the latent action future prediction model are both able to reconstruct the background and the vehicles perfectly. At the same time, the one action future prediction model fails to reconstruct the agent accurately, whereas the latent action future prediction model is able to reconstruct the agent almost perfectly. This provides evidence for the effectiveness of our proposed latent action mining approach at discovering pure action groundings.

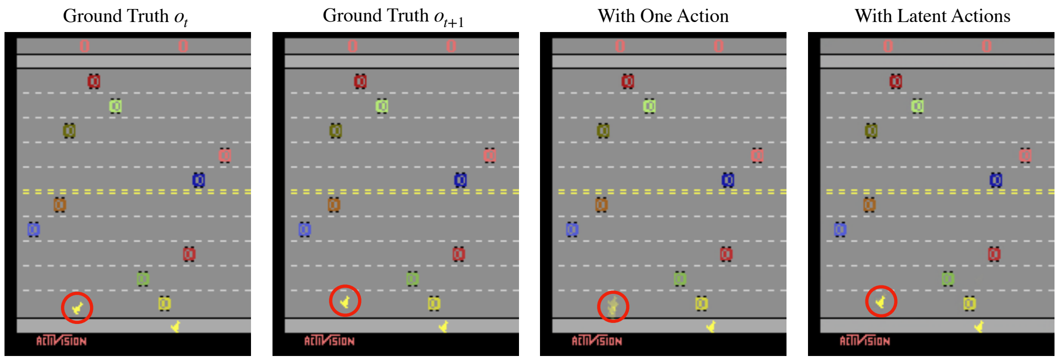

Figure S8: We visualize the Freeway future prediction models' reconstructions for o_{t+1}. From left to right, the ground truth o_t, the ground truth o_{t+1}, the reconstruction for o_{t+1} by the future prediction model with one action, the reconstruction for o_{t+1} by the future prediction model with latent actions. The agent is circled in each image.

Furthermore, we visualize the learned latent actions for Freeway in Figure S9. In Freeway, the agent can only move along the y-axis, and consequently, the environment action space only has three actions: move up, move down, and no-op. This means that the agent's y-displacement between the current frame and the previous frame directly corresponds to the ground truth action taken. In this visualization, we visualize the the chosen latent action (by color: blue, orange, or green) as a function of the agent's y-position in the current frame (x-axis) and the y-displacement relative to the previous frame (y-axis). Note that as mentioned in the paper, we learn the value functions over the top three most dominant actions to stabilize training; for this reason, the visualization only consists of three latent actions.

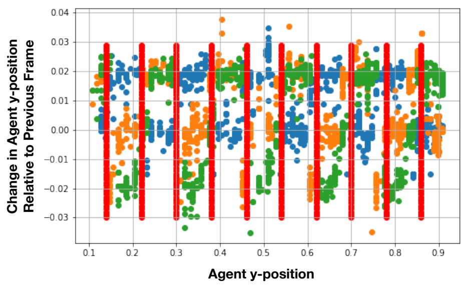

Figure S9: Visualization of the latent actions learned in Freeway. X-axis is the agent's y-position, y-axis is the displacement of the agent (which is indicative of ground truth action).

Within each vertical region marked by the red bars, we see that the latent actions are split into distinct clusters based on the agent's y-displacement. Because the agent's y-displacement directly corresponds to the ground truth action taken, this indicates that given the agent's y-position, the learned latent actions encode information about the ground truth action. Hence, we qualitatively confirm that the state-based purity of the latent actions is high.

A.8 Value Function Details

Value Function Model Selection The numbers reported in Table 1 are the 95th percentile Spearman's correlation coefficients over the course of training. If training is stable and converges, this corresponds to taking the final value, and in the case that training is not stable and diverges, this acts as a form of early stopping. We take the 95th percentile as opposed to the maximum to eliminate outliers.

Below we present visualizations of value functions learned from LAQ.

Freeway. We visualize the value functions learned using our latent actions for Freeway. Figure S10 plots the values over the course of an episode. In Freeway, the agent has to move vertically up and down the screen to cross a busy freeway, receiving reward when it successfully gets to the other side. In a single episode, the agent can cross the freeway multiple times; each time the agent makes it to the other side, the agent's location is reset to the original starting location, allowing the agent to attempt to cross the freeway once again. For this reason, we see the value increase as the agent gets closer to the other side of the road, and then drop as soon as its position resets to the starting location. As evident, the peaks of the learned value function correspond highly to the environment reward.

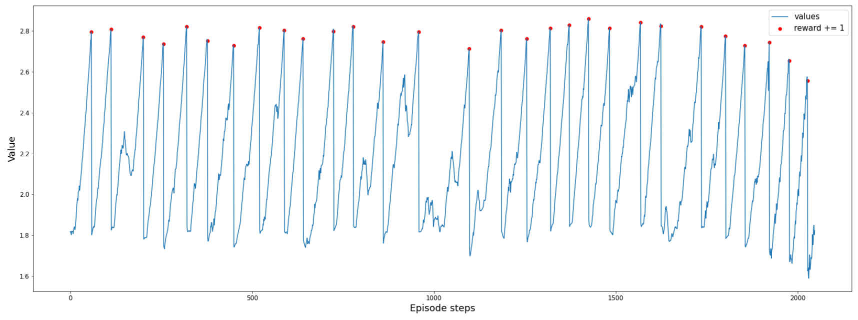

Figure S10: We visualize the value predicted by our learned value function over the course of one episode of Freeway. The red points correspond to when the agent receives a reward from the environment.

3D Visual Navigation. Figure S11 visualizes the learned value map in the 3D Visual Navigation branching environment from [8]. As depicted in Figure 6, there are two goals: G_{near} and G_{far}. The figure on the right illustrates that the value function learned by utilizing our latent actions correctly assigns high value to the regions surrounding the two goal locations, and low value elsewhere. Additionally, we learn the value functions with DQN using a dataset obtained by using a sub-optimal policy which prefers to go to the goal further away rather than the goal close by. Despite this sub-optimality, the learned value function correctly assigns a higher value to the nearby goal than the goal which is further away.

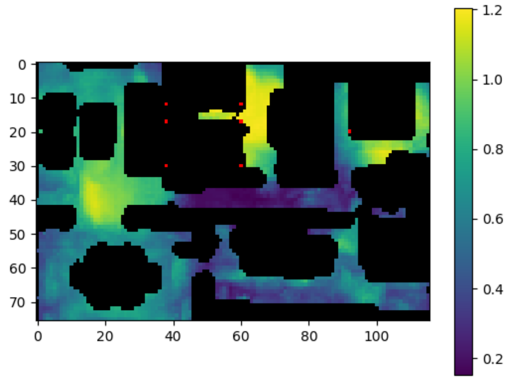

Figure S11: Visualization of the learned value function for 3D Visual Navigation.

2D Continuous Control. Figure S12 shows a visualization of the value function learned using latent actions for the 2D Continuous Control environment. While the observation space of this environment is (x, y, v_x, v_y) *i.e.* location and velocity of the agent, we produce this visualization over just the location of the agent. Based on just the location of the agent, the optimal value function would be a monotonically decreasing function as the distance from the goal location increases. In this particular environment, the goal of the agent is to get to the top-right corner of the maze. The visualization shows that the learned value function produces high values around this goal region at the top-right corner of the maze, and gradually lower values the farther away you go. This figure visually is in line with our quantitative results in Table 1 which show that the value function learned using latent actions in the 2D Continuous Control environment highly correlates to that learned using ground truth actions.

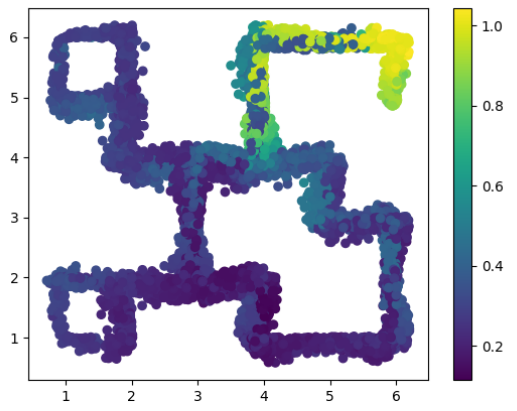

Figure S12: Visualization of the learned value function for 2D Continuous Control.

Kitchen Manipulation. We visualize the values as a function of both the angle that the microwave door makes with respect to its starting state as well as the distance between the end-effector and the microwave door handle in Figure S13. The task here is to open the microwave door: the agent receives a binary reward when the angle that the microwave door makes with respect to its starting state (i.e. closed microwave door) is above a threshold (approx. 0.7 radians), and zero otherwise. As expected, we see that the predicted value of a state increases as the angle of the microwave door increases. The end-effector position does not start out at the microwave door handle, but rather is initialized a fixed distance away from the handle. As a result, the agent must first move the end-effector to the microwave door handle, before interacting with the door handle to open it. In addition to this, we hypothesize that as the distance between the end-effector and the microwave door handle decreases, the predicted value should increase. While not as obvious as the relation between the microwave door angle and the value, we see some indication that as the distance between the end-effector and the microwave door handle decreases, the value increases.

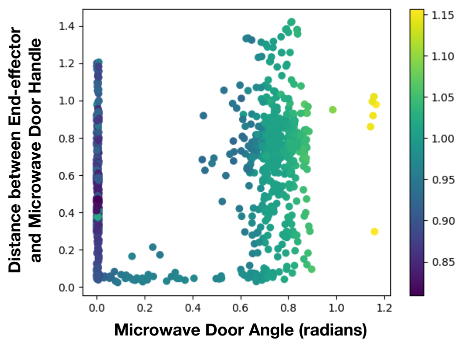

Figure S13: Visualization of the learned value function in Kitchen Manipulation. The x-axis is the microwave angle in radians, the y-axis is the distance between the end-effector and the microwave door handle, with the colors corresponding to the magnitude of the values.

A.9 LAQ in Stochastic MDPs

We have conducted experiments on stochastic MDPs in the gridworld environment. Specifically, we do this by adding sticky actions, such that with a certain probability (called the stickiness parameter), the environment executes the previous action as opposed to the given action. We run LAQ with data from this stochastic environment and examine a) purity of latent actions, b) quality of learned value functions, and c) correctness of implied behavior.

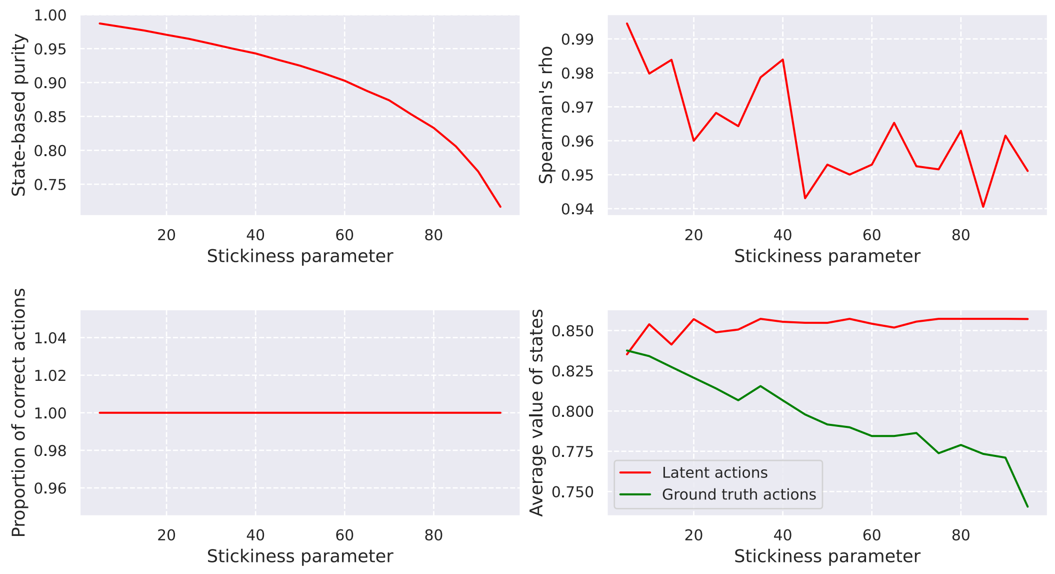

Figure S14: As a function of stochasticity in environment, we plot: state-based purity of the latent actions (top left), Spearman's rank correlation between LAQ learned value function and ground truth value function (top right), correctness of implied behavior by the LAQ learned value function (bottom left), and average state-value of LAQ learend value function, and the ground truth value function (bottom right).

As expected, Figure S14 (left) shows that the state-based purity of the learned latent actions falls off as the stochasticity (stickiness) increases. However, these impure latent actions have little effect on the Spearman's rank correlation (top right). The Spearman's rank correlation does decrease but remains reasonably high even when the stickiness parameter is set to 95%. While the ordering of the values of the different states doesn't change by much, we note increasing amount of over-estimation in the values (bottom right). This is expected as with increasing stochasticity, there is an increasingly large gap between what LAQ thinks it can control, and what it can actually control. However, the behavior implied by LAQ is still always correct (bottom left), as the over-estimation is uniform over all states. Thus, it is possible to get good performance from LAQ in some stochastic environments. However, it is possible to also construct simple scenarios where LAQ, and for that matter any deterministic algorithm, will suffer as we describe in Section A.2.

A.10 Algorithm

Algorithm 1 LAQ

```
 1: Given dataset $D$ of $(o_t, o_{t+1}, r_t)$ triples
 2: for each epoch do                                                    ▷ Latent Action Mining
 3:     for sampled batch $(o_t, o_{t+1}, r_t) \sim D$ do
 4:         $L(o_t, o_{t+1}) = \min_{\hat{a} \in \hat{\boldsymbol{A}}} \; l(f_\theta(o_t, \hat{a}), o_{t+1})$
 5:         $\theta \leftarrow \theta - \alpha \nabla_\theta L(o_t, o_{t+1})$            ▷ Section 4.1
 6:     end for
 7: end for
 8: let $\hat{g}(o_t, o_{t+1}) = \arg\min_{\hat{a} \in \hat{\boldsymbol{A}}} l(f_\theta(o_t, \hat{a}), o_{t+1})$
 9: $\hat{D} = \{(o_t, o_{t+1}, r_t, \hat{g}(o_t, o_{t+1})) \mid (o_t, o_{t+1}, r_t) \in D\}$     ▷ Latent Action Labeling
10: for each epoch do
11:     for sampled batch $(o_t, o_{t+1}, r_t, \hat{a}) \sim \hat{D}$ do          ▷ Q-learning with Latent Actions
12:         Q-learning update to learn $Q(s, \hat{a})$
13:     end for
14: end for
15: $V(s) = \max_{\hat{a} \in \hat{\boldsymbol{A}}} \; Q(s, \hat{a})$
```