Non-Markovian Reward Modelling from Trajectory Labels via Interpretable Multiple Instance Learning

Joseph Early [*†§] **Tom Bewley** [*‡§] **Christine Evers** [†] **Sarvapali Ramchurn** [†§]

Abstract

We generalise the problem of reward modelling (RM) for reinforcement learning (RL) to handle non-Markovian rewards. Existing work assumes that human evaluators observe each step in a trajectory independently when providing feedback on agent behaviour. In this work, we remove this assumption, extending RM to capture temporal dependencies in human assessment of trajectories. We show how RM can be approached as a multiple instance learning (MIL) problem, where trajectories are treated as bags with return labels, and steps within the trajectories are instances with unseen reward labels. We go on to develop new MIL models that are able to capture the time dependencies in labelled trajectories. We demonstrate on a range of RL tasks that our novel MIL models can reconstruct reward functions to a high level of accuracy, and can be used to train high-performing agent policies.

1 Introduction

There is growing consensus around the view that aligned and beneficial AI requires a reframing of objectives as being contingent, uncertain, and learnable via interaction with humans [35]. In reinforcement learning (RL), this proposal has found one formalisation in reward modelling (RM): the inference of agent objectives from human preference information such as demonstrations, pairwise choices, approval labels, and corrections [29]. Prior work in RM typically assumes that a human evaluates the *return* (quality) of a sequential trajectory of agent behaviour by summing equal and independent *reward* assessments of instantaneous states and actions, with the aim of RM being to reconstruct the underlying reward function. However, in reality the human's experience of a trajectory is likely to be temporally extended (e.g., via a video clip [12] or real-time observation), which opens the door to dependencies between earlier events and the assessment of later ones. The independence assumption may be both psychologically unrealistic given human memory limitations [26], and technically naïve given the difficulty of building complete instantaneous state representations [25]. We thus seek to generalise RM to allow for temporal dependencies in human evaluation, by postulating *hidden state* information that accumulates over a trajectory. Reconstruction of the human's preferences now requires the modelling of hidden state dynamics alongside the reward function itself.

In tackling this generalised problem, we identify a structural isomorphism between RM (specifically from trajectory return labels) and the established field of multiple instance learning (MIL) [10]. Trajectories are recast as *bags* and constituent state-action pairs as *instances*, which collectively contribute to labels provided at the bag level by interacting in potentially complex ways. This mapping inspires a range of novel MIL model architectures that use long short-term memory (LSTM) modules [19] to recover the hidden state dynamics, and learn instance-level reward predictions from return-labelled trajectories of arbitrary length. In experiments with synthetic oracle labels, we show that our MIL RM models can accurately reconstruct ground truth hidden states and reward functions for non-Markovian tasks, and can be straightforwardly integrated into RL agent training to achieve performance matching, or even exceeding, that of agents with direct access to true hidden states and rewards. We then apply interpretability analysis to understand what the models have learnt.

[*]Equal contribution
[†]University of Southampton, United Kingdom; {J.A.Early,C.Evers,sdr1}@soton.ac.uk
[‡]University of Bristol, United Kingdom; tom.bewley@bristol.ac.uk
[§]The Alan Turing Institute, United Kingdom

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Our contributions are as follows:

1. We generalise RM to handle *non-Markovian* rewards that depend on hidden features of the environment or the psychology of the human evaluator in addition to visible states/actions.
2. We identify a structural connection between RM and MIL, creating the opportunity to transfer concepts and methods between the two fields.
3. We propose novel LSTM-based MIL models for this generalised RM problem, and develop interpretability techniques for understanding and verifying the learnt reward functions.
4. We compare our proposed models to existing MIL baselines on five non-Markovian tasks, evaluating return prediction, reward prediction, robustness to label noise, and interpretability.
5. We demonstrate that the hidden state and reward predictions of our MIL RM models can be used by RL agents to solve non-Markovian tasks.

The remainder of this work is as follows. Section 2 discusses related work in RM and MIL, Section 3 gives a formal problem definition and describes our MIL-inspired methodology, and Section 4 presents our experiments and results. We discuss key findings in Section 5, and Section 6 concludes. All of our source code is available in a public repository.[1]

2 Background and Related Work

Reward Modelling RM [29] aims to infer a reward function from revealed human preference information such as demonstrations [32], pairwise choices [12], corrections [4], good/bad/neutral labels [33], or combinations thereof [24]. Most prior work assumes a human evaluates a trajectory by summing independent rewards for each state-action pair, but in practice their experience is likely to be temporally extended (e.g., via a video clip), creating the opportunity for dependencies to emerge between earlier events and the assessment of later ones. As noted by Chan et al. [11] and Bewley and Lecue [8], such dependencies may arise from cognitive biases such as anchoring, prospect bias, and the peak-end rule [26], but they could equally reflect rational drivers of human preferences not captured by the state representation. Some efforts have been made to model temporal dependencies, such as a discrete psychological mode which evolves over consecutive queries about hypothetical trajectories [6], or a monotonic bias towards more recently-viewed timesteps due to human memory limitations [28]. Elsewhere, Shah et al. [36] use human demonstrations and binary approval labels to learn temporally extended task specifications in logical form. In comparison to these restricted examples, our work provides a more general approach to capturing temporal dependencies in RM.

Non-Markovian Rewards In the canonical RL problem setup of a Markov decision process (MDP), rewards depend only on the most recent state-action pair. In a non-Markovian reward decision process (NMRDP) [2], rewards depend on the full preceding trajectory [2]. NMRDPs can be *expanded* into MDPs (and thus solved by RL) by augmenting the state with a hidden state that captures all reward-relevant historical information, but this is typically not known *a priori*. Data-driven approaches to learning NMRDP expansions [21] often make use of domain-specific propositions and temporal logic operators [3, 39, 41]. Outside of the RM context, recurrent architectures such as LSTMs have been used in NMRDPs to reduce reliance on pre-specified propositions [23]. They also have a long history of use in partially observable MDPs, where dynamics are also non-Markovian [5, 17, 46].

Multiple Instance Learning In MIL [10], datasets are structured as collections of bags $X_i \in \mathbf{X}$, each of which is comprised of instances $\{x_1^i, \ldots, x_k^i\}$ and has an associated bag-level label Y_i and instance-level labels $\{y_1^i, \ldots, y_k^i\}$. The aim is to construct a model that learns solely from bag labels; instance labels are not available during training, but may be used later to evaluate instance-level predictions. The simplest MIL approaches assume that instances are independent and that the bag is unordered, but models exist for capturing various types of instance dependencies [22, 42, 45]. LSTMs have emerged as a natural architecture for modelling temporal dependencies among ordered bags, where they can be utilised to aggregate instance information into an overall bag representation. They have previously been applied to standard MIL benchmarks [44], as well as specific problems such as Chinese painting image classification [30]. As we discuss in Section 3.2, these existing models are somewhat unsuitable for use in RM, leading us to propose our own novel model architectures.

[1] https://github.com/JAEarly/MIL-for-Non-Markovian-Reward-Modelling

3 Methodology

In this section, we present the core methodology of our work. We formally define the new paradigm of non-Markovian RM (Section 3.1), before drawing on the MIL literature to propose models that can be used to solve this generalised problem (Section 3.2). We then go on to discuss how we can use our learnt RM models for training RL agents on non-Markovian tasks (Section 3.3).

3.1 Formal Definition of Non-Markovian RM

Consider an agent interacting with an environment with Markovian dynamics. At discrete time t, the current environment state $s_t \in \mathcal{S}$ and agent action $a_t \in \mathcal{A}$ condition the next environment state s_{t+1} according to the dynamics function $D : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$. A trajectory $\xi \in \Xi$ is a sequence of state-action pairs, $\xi = ((s_0, a_0), ..., (s_{T-1}, a_{T-1}))$, and a human's preferences about agent behaviour respect a real-valued return function $G : \Xi \to \mathbb{R}$. In traditional (Markovian) RM, return is assumed to decompose into a sum of independent rewards over state-action pairs, $G(\xi) = \sum_{t=0}^{T-1} R(s_t, a_t)$, and the aim is to reconstruct $R' \approx R$ from possibly-noisy sources of preference information. In our generalised non-Markovian model, we consider the human to observe a trajectory sequentially and allow for the possibility of hidden state information that accumulates over time and parameterises R:

$$G(\xi) = \sum_{t=0}^{T-1} R(s_t, a_t, h_{t+1}) \quad \text{where} \quad h_{t+1} = \delta(h_t, s_t, a_t), \tag{1}$$

δ is a hidden state dynamics function, and h_0 is a fixed value for the initial hidden state. Reconstruction of the human's preferences now requires the estimation of $\delta' \approx \delta$ and $h'_0 \approx h_0$ alongside $R' \approx R$. We visualise the difference between Markovian and non-Markovian RM in Figure 1.

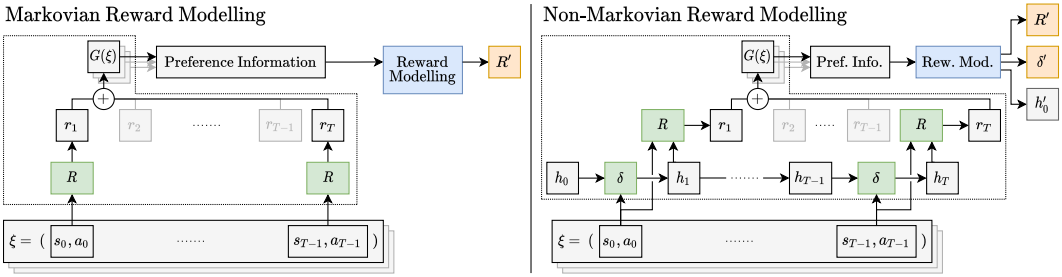

Figure 1: In Markovian RM, the human is assumed to sum $(+)$ over independent and equal reward assessments for the state-action pairs in a trajectory. In non-Markovian RM, per-timestep rewards additionally depend on hidden state information h that accumulates over time.

The hidden state h may be interpreted as (1) an external feature of the environment that is detectable by the human but excluded from the state, or (2) a psychological feature of the person themselves, through which their response to each new observation is influenced by what they have seen already. The latter framing is more interesting for our purposes, and connects to the psychological literature on human judgement, memory, and biases [26]. In practice, hidden state information may encode the human's preferences about the order in which a sequence of behaviours should be performed, the effect of historic observations on their subjective mood (and in turn on their reward evaluations), or cognitive biases which corrupt the way they aggregate instantaneous rewards into trajectory-level feedback. All of these complications are liable to arise in practical RM applications, but cannot be handled when the Markovian reward assumption is made. Appendix B elaborates on this discussion, presenting motivating use cases and limitations of non-Markovian RM.

In this work, we focus on one of the simplest and most explicit forms of preference information: direct labelling of returns $G(\xi_i)$ for a dataset of N trajectories $\{\xi_i\}_{i=1}^N$. We aim to solve the reconstruction problem by minimising the squared error in predicted returns:

$$\underset{R', \delta', h'_0}{\text{argmin}} \sum_{i=1}^{N} \left(G(\xi_i) - \sum_{t=0}^{T_i-1} R'(s_{i,t}, a_{i,t}, h'_{i,t+1}) \right)^2 \text{ where } \begin{array}{l} h'_{i,0} = h'_0 \\ h'_{i,t+1} = \delta'(h'_{i,t}, s_{i,t}, a_{i,t}) \end{array} \forall i \in \{1..N\}. \tag{2}$$

We observe that Equation 2 perfectly matches the definition of a MIL problem. Each trajectory ξ_i can be considered as an ordered bag of instances $((s_{i,0}, a_{i,0}), ..., (s_{i,T_i-1}, a_{i,T_i-1}))$ with unobserved instance labels $R(s_{i,t}, a_{i,t}, h_{i,t+1})$, an observed bag label $G(\xi_i) = \sum_{t=0}^{T-1} R(s_{i,t}, a_{i,t}, h_{i,t+1})$, and temporal instance interactions via the changing hidden state $h_{i,t}$. This correspondence motivates us to review the space of existing MIL models (specifically those that model temporal dependencies among instances) to provide a starting point for developing our non-Markovian RM approach.

3.2 MIL RM Architectures

The MIL literature contains a variety of architectures for handling temporal instance dependencies, including graph neural networks (GNNs) [42] and transformers [37]. While effective for many problems, such architectures are an unnatural fit to non-Markovian RM as they contain no direct analogue of a hidden state h' carried forward in time, instead handling dependencies via some variant of message-passing between instances. LSTM-based MIL architectures [30, 44] provide a more promising starting point since they explicitly represent both h' (implemented as a continuous-valued vector) and its temporal dynamics function δ' (a particular arrangement of gating functions).

Starting from an existing LSTM-based MIL architecture, we propose two successive extensions as well as a naïve baseline that cannot handle temporal dependencies. All four architectures include a feature extractor (FE) for mapping state-action pairs into feature vectors and a head network (HN) that outputs predictions. These architectures are depicted in Figure 2. Note we use the same nomenclature as [10] and [45]: *embedding space* approaches produce an overall bag representation that is used for prediction, while *instance space* approaches produce predictions for each instance in the bag and then aggregate those predictions to a final bag prediction.

Base Case: Embedding Space LSTM This architecture, proposed by Wang et al. [44], processes all instances in a bag sequentially and uses the final LSTM hidden state as a bag embedding. This is fed into the HN, which predicts the bag label g' (return in the RM context). Although this model can account for temporal dependencies, it does not inherently produce instance predictions (rewards), which require some post hoc analysis to recover. While methods exist for computing instance importance values as a form of interpretability [13], these are not guaranteed to sum to the bag label as stipulated by the reward-return formulation. We propose a new method: at time t, the predicted reward r'_t is calculated by feeding the LSTM hidden state at times $t-1$ and t into the HN to obtain two *partial* bag labels/returns g'_{t-1} and g'_t, and computing the difference of the two, i.e., $r'_t = g'_t - g'_{t-1}$. We define $g'_0 = 0$. This post hoc computation is shown in purple in Figure 2.

Extension 1: Instance Space LSTM The post hoc computation of reward proposed above is rather inelegant and often yields poor predictions (see Section 4 and Appendix D), likely because rewards are never computed or back-propagated through during learning. This leads us to propose an improved architecture, which is structurally similar but differs in how network outputs are mapped onto RM concepts. The change places reward predictions on the back-propagation path. Given the LSTM hidden state at time t, the output of the HN is taken to be the instantaneous reward r'_t rather than the partial return. Rewards are computed sequentially for all timesteps in a trajectory and summed to give the return prediction g'. We thereby obtain a model that both handles temporal dependencies and produces explicitly-learnt reward predictions.

Extension 2: Concatenated Skip Connection (CSC) Instance Space LSTM In both of the preceding architectures, the LSTM hidden state h'_t is the sole input to the HN. This requires h'_t to represent all reward-relevant information from both the true hidden state h_t and the latest state-action pair s_{t-1}, a_{t-1} to achieve good performance. To lighten the load on the LSTM, we further extend the Instance Space LSTM model with a skip connection [18, 20] which concatenates the FE output onto the hidden state before feeding it to the HN. In principle, this should allow the hidden state to solely focus on representing temporal dependencies. As well as improving RM performance compared to an equivalent model without skip connections, we find in Section 5.1 that this modification tends to yield more interpretable and disentangled hidden state representations.

Markovian Baseline: Instance Space Neural Network (NN) To quantify the cost of ignoring temporal dependencies, we also run experiments with a baseline architecture that feeds only the FE output for each state-action pair into the HN, yielding fully-independent reward predictions which are summed to give the return prediction. This independent predict-and-sum architecture has precedence in both MIL, where it is referred to as mi-Net by Wang et al. [45], and in RM, where it embodies the de facto standard Markovian reward assumption [12].

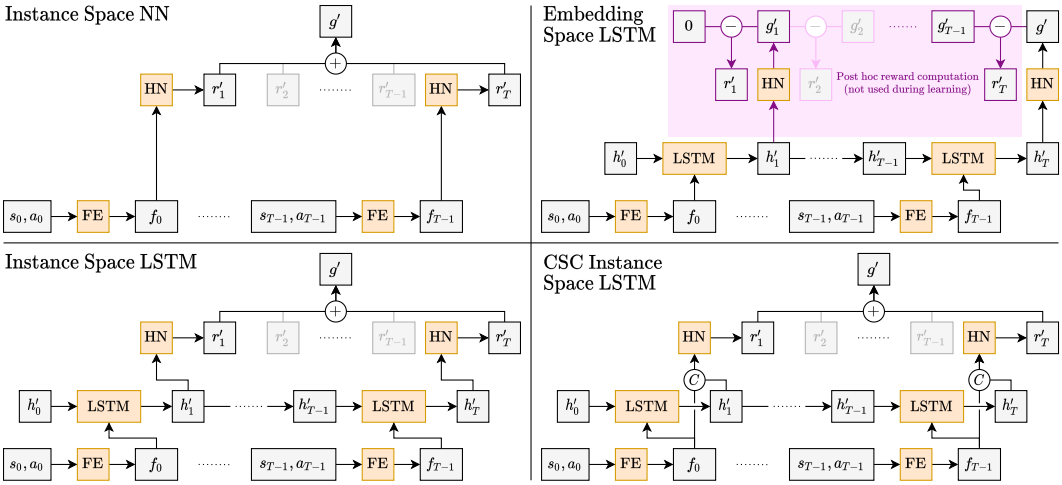

Figure 2: MIL architectures used in this work. FE = feature extractor; HN = head network; $(+)$ = scalar summation; $(-)$ = scalar subtraction; (C) = vector concatenation.

3.3 Training Agents with Non-Markovian RM Models

In this work, as in RM more widely, we are not solely interested in learning reward functions to represent human preferences, but also in the downstream application of rewards to train agents' action-selection policies. After optimising our LSTM-based models on offline trajectory datasets, we deploy them at the interface between conventional RL agents and their environments. Going beyond prior work, where a learnt model is used to either generate a reward signal for an agent to maximise [12] or augment its observed state representation with hidden state information [21], our models serve a dual role, providing *both* rewards and state augmentations. Figure 3 describes this setup in detail.

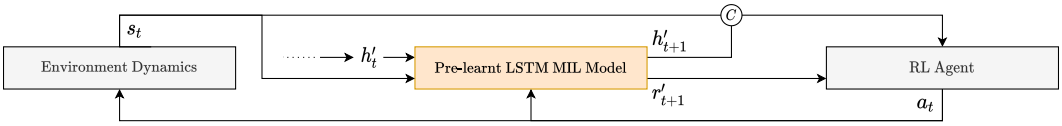

Figure 3: During RL agent training, our LSTM MIL models sit at the centre of the agent-environment loop by which states s_t and actions a_t are exchanged. We focus on episodic tasks, where the environment state periodically resets. The LSTM hidden state is simultaneously reset to h'_0 at the start of an episode, then is iteratively updated over time t given the state-action pairs s_t, a_t. At time t the environment state s_t is augmented with the post-update hidden state h'_{t+1} by concatenation, and this augmented state is observed by the agent. s_t, a_t and h'_{t+1} are used to compute a reward r'_{t+1} following the relevant steps from Figure 2, and the reward is also sent to the agent. In the language of NMRDPs, the hidden state augmentation *expands* the agent's learning problem into an MDP by providing the additional information required to make the rewards Markovian. Note that unlike during learning of the MIL models, return predictions are never required.

4 Experiments and Results

After initially validating our models on several toy datasets (see Appendix D), we focus the bulk of our evaluation on five RL tasks. As running experiments with people is costly, we use the standard RM approach of generating synthetic preference data (here trajectory return labels) using ground truth *oracle* reward functions [12] (for a discussion comparing the use of oracle and human labels, see Appendix C). Unlike prior work, these oracle reward functions depend on historical information that cannot be recovered from the instantaneous environmental state, thereby emulating the disparity between the information that a human evaluator possesses while viewing a trajectory sequentially, and that contained in the state alone. In this section, we introduce our RL tasks (Section 4.1), evaluate the quality of reward reconstruction (Section 4.2), investigate the use of MIL RM models for agent training (Section 4.3), and evaluate their robustness to label noise (Section 4.4).

4.1 RL Task Descriptions

We apply our methods to five non-Markovian RL tasks, the first four of which are within a common 2D navigation environment and are specifically designed to capture different kinds of non-Markovian structure. Each environment has two spawn zones and an episode time limit of $T = 100$; see Figure 4. In each case, the environment state contains the x, y position of the agent only. The tasks involve moving into a *treasure* zone, contingent on some hidden information that cannot be derived from the current x, y position, but is instead a function of the full preceding trajectory. In the first two cases the hidden information varies with time only, but in the other two it depends on the agent's past positions.

Timer For times $t \leq 50$ the treasure gives a reward of -1 for each timestep that the agent spends inside it, before switching to $+1$ thereafter. Since time is not included in the environment state, recovering the reward function by only observing the agent's current position is impossible.

Moving The Timer task only captures a binary change, therefore we generalise it to be continuous. In this case, the treasure zone oscillates left and right at a constant speed. Again, this is not captured in the environment state, but can be recovered if the length of the preceding trajectory is known.

Key Before reaching the treasure zone, the agent must first enter a second zone to collect a key; otherwise it receives 0 reward. As the key's status is not captured in the environment state, a temporal dependency exists between the agent's past positions and the reward it obtains from the treasure.

Charger We generalise the Key task by replacing the key zone with a charging zone that builds up the amount of reward the agent will receive when it reaches the treasure. The reward now depends not only on whether the agent visits a zone (binary), but how long it spends there (continuous).

For the fifth and most complex task, we adapt **Lunar Lander** from OpenAI Gym [9], adding the condition that the lander should take off again and stably hover after 50 timesteps on the landing pad. This is analogous to the Charger task but with a larger state-action space and longer episodes ($T = 500$). Further details on the tasks and MIL model hyperparameters are given in Appendix E.

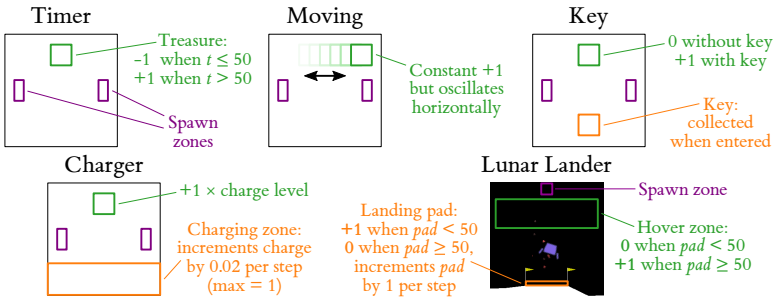

Figure 4: Visualisations of the five non-Markovian RL tasks.

An important design decision for the LSTM-based models is the size of the hidden state, as it affects both performance and interpretability. For all the above tasks, we know *a priori* that it is possible to capture the temporal dependencies in at most two dimensions, so we constrain our models to use 2D hidden states. This allows us to visualise and interpret the hidden representations in Section 5.1.

4.2 Reward Modelling Results

Below we discuss the performance of the reward reconstruction for the different MIL RM models on our five RL tasks. For each task, we generate initial trajectories to form our MIL RM datasets (see Appendix E). Results from MIL models trained on these trajectories are given in Table 1. We observe that the CSC Instance Space LSTM model is on average the best-performing model for predicting both trajectory returns and timestep rewards. While the Embedding Space LSTM model performs best at predicting return on the Key and Lunar Lander tasks (as is not constrained to the summation of reward predictions as in the other architectures), it struggles on the reward metric (due to the use of a proxy post hoc method). As it is important for these models to achieve strong performance on both return and reward prediction, the Instance Space LSTM and CSC Instance Space LSTM models are better candidates than the Embedding Space LSTM. Also note that the Instance Space NN that serves as our Markovian RM baseline performs very poorly on return prediction, indicating that these tasks indeed cannot be learnt without modelling temporal dependencies.

Table 1: MIL RM return (top) and reward (bottom) results, using ten repeats. The Lunar Lander results are the average test set MSE of the top five models (with scaling; see Appendices E.4 and E.5). For the other tasks, each measurement is the test set MSE averaged over all ten repeats. The standard errors of the mean are given, and the Lunar Lander reward results are scaled by 1×10^{-5}.

Model	Timer	Moving	Key	Charger	Lunar Lander
Instance Space NN	130.8 ± 1.530	22.24 ± 0.441	7.764 ± 0.232	7.783 ± 0.214	2.297 ± 0.058
Embedding Space LSTM	3.151 ± 0.662	13.04 ± 0.899	$\mathbf{0.360} \pm \mathbf{0.055}$	0.689 ± 0.124	$\mathbf{0.416} \pm \mathbf{0.048}$
Instance Space LSTM	7.313 ± 2.627	11.13 ± 1.169	0.488 ± 0.062	0.628 ± 0.126	1.223 ± 0.431
CSC Instance Space LSTM	$\mathbf{0.605} \pm \mathbf{0.166}$	$\mathbf{5.307} \pm \mathbf{0.299}$	0.391 ± 0.083	$\mathbf{0.125} \pm \mathbf{0.012}$	0.501 ± 0.035
Instance Space NN	0.217 ± 0.001	0.068 ± 0.000	0.011 ± 0.000	0.025 ± 0.000	7.484 ± 0.861
Embedding Space LSTM	101.8 ± 60.35	3.033 ± 0.715	0.010 ± 0.008	0.037 ± 0.016	120.2 ± 24.27
Instance Space LSTM	0.263 ± 0.038	0.069 ± 0.005	0.002 ± 0.000	0.005 ± 0.001	9.336 ± 3.116
CSC Instance Space LSTM	$\mathbf{0.073} \pm \mathbf{0.016}$	$\mathbf{0.026} \pm \mathbf{0.002}$	$\mathbf{0.001} \pm \mathbf{0.000}$	$\mathbf{0.001} \pm \mathbf{0.000}$	$\mathbf{7.365} \pm \mathbf{1.032}$

4.3 RL Training Results

Following the method in Section 3.3, we then train Soft Actor-Critic [15] (Lunar Lander) and Deep Q-Network [31] (all others) RL agents to optimise the rewards learnt by the LSTM-based models. We evaluate agent performance in a post hoc manner by passing its trajectories to the relevant oracle. This evaluation provides an end-to-end measure of both reward reconstruction and policy learning, and is standard in RM [12]. We baseline against agents trained with access to: a) the oracle reward function and the oracle hidden states, and b) just the oracle reward function without hidden states (i.e, using only the environment states that are missing information). In Figure 5, we observe that the CSC Instance Space LSTM model enables the best RL agent performance, coming closest to the oracle. Interestingly, for the Timer and Lunar Lander tasks, the CSC Instance Space LSTM model actually outperforms the use of the oracle, suggesting that the learnt hidden states are easier to exploit for policy learning than the raw oracle state (we investigate what these models have learnt in Section 5.1). Note the poor performance of agents trained without hidden state information, which aligns with expectations. For further details on agent training, see Appendix F.

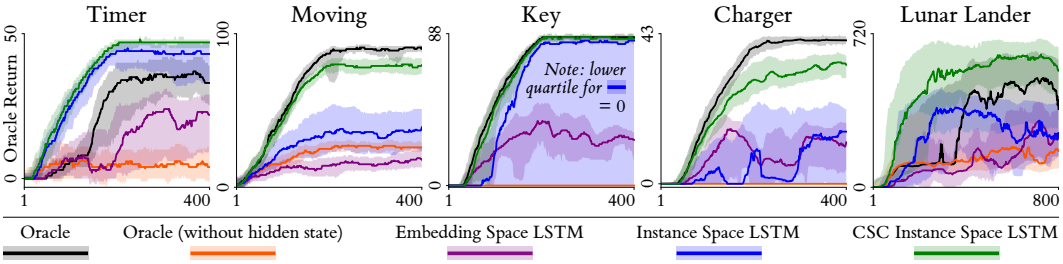

Figure 5: RL performance for different training configurations on our five RL tasks. The results given are the medians and interquartile ranges. For the oracle results, we trained ten repeats, and for the MIL-LSTM results, we performed one RL training run for each MIL-LSTM model repeat.

For Lunar Lander, we perform a deeper analysis of RL training performance, by decomposing the oracle return curves from Figure 5 into the four reward components R_{pad}, $R_{\text{no_contact}}$, R_{hover} and R_{shaping} (see Appendix E.1 for definitions). The decomposed curves, shown in Figure 6, allow us to diagnose the origins of the performance disparity between runs using different LSTM model architectures. There is relatively little separation in performance on the shaping reward R_{shaping} and pad contact reward R_{pad} (for the latter, all runs end up reliably achieving the maximum possible reward of 49, although those using Embedding Space LSTM models require significantly more training time). This suggests that all models have been able to recover these components with reasonable fidelity. However, there are marked differences in performance on $R_{\text{no_contact}}$ and R_{hover} (the components relating to the second task stage of taking off and moving to the hover zone). For R_{hover}, runs using the CSC Instance Space LSTM peak at a return of around 200 from this component, while those using the other two models almost never achieve non-zero return, i.e., only the RL agents trained using the CSC Instance Space LSTM RM models reliably learn to hover. This indicates that the models have learnt very different representations of reward and hidden state dynamics, which are effective for policy learning in the case of the CSC model, and highly ineffective for the others.

Observe that runs using CSC Instance Space LSTM models outperform those with direct access to the ground truth oracle on all components, and most markedly on R_{hover}. This counterintuitive finding suggests that this model reliably learns hidden state representations that are easier for RL agents to leverage for policy learning than the ground truth ones, and potentially that certain errors in the reward prediction may actually be beneficial for the purpose of helping agents to complete the underlying task (especially the hovering stage). In typical RL parlance: the model's reward function appears to be better *shaped* than the ground truth. The potential origins of this better-than-oracle phenomenon are investigated in Figure A5 (Appendix G).

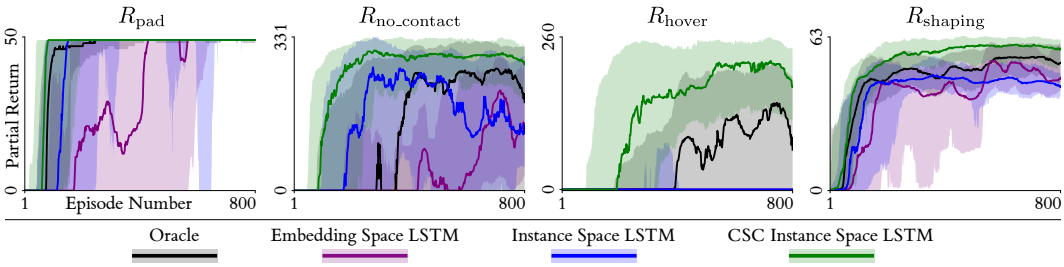

Figure 6: Decomposed oracle return curves for Lunar Lander.

4.4 Robustness to Mislabelling

In this work, the return labels are provided by oracles rather than real people. When using human evaluators, there is likely to be uncertainty in the labels, and it is important to evaluate model robustness against such noise [28]. We implement noise through label swapping [34]; this ensures the marginal label distribution remains the same and does not include out-of-distribution returns. In Figure 7, we show how both return and reward prediction decay with noise levels increasing from 0 (no labels swapped) to 0.5 (half swapped). The rate, smoothness, and consistency (across three repeats) of this decay varies between tasks, with decays in return prediction generally being smoother. We observe that the CSC Instance Space LSTM model remains the strongest predictor of both return and reward in the majority of cases, indicating general robustness and providing evidence that the model should still be effective with imperfect human labels. On all metrics aside from Timer reward loss (where the mix of negative and positive rewards makes the effect of noise especially unpredictable), a noise level of at least 0.3 is required for the CSC Instance Space LSTM model to perform as badly as the Instance Space NN baseline does with no noise at all.

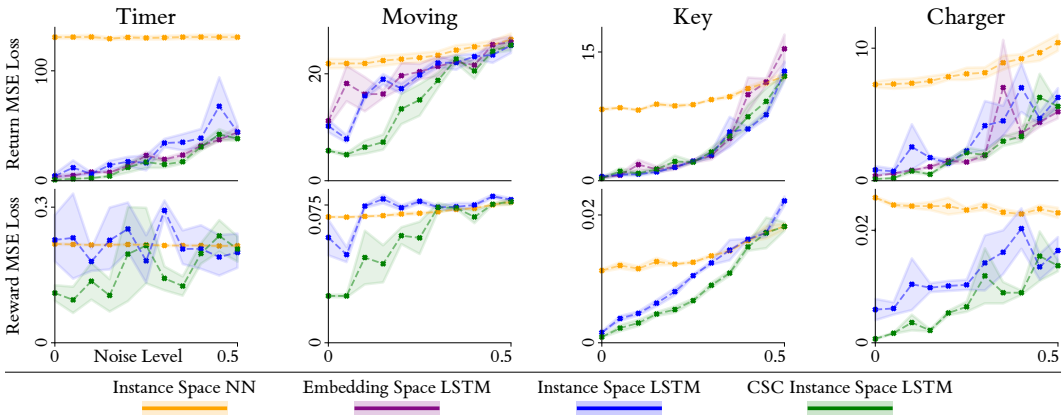

Figure 7: Performance of MIL RM models subject to label noise. We omit the Embedding Space LSTM reward losses as they are very high, and the Lunar Lander task due to long training times.

5 Discussion

In this section, we seek to interpret our MIL RM models, analysing the distribution of learnt hidden states (Section 5.1) as well as their temporal dynamics over the course of a trajectory (Section 5.2). Finally, in Section 5.3 we discuss the limitations of this work and potential areas for future work.

5.1 Hidden State Analysis

The primary purpose of RM is to perform accurate reward reconstruction to facilitate agent training, but there is a secondary opportunity to improve understanding of human preferences through interpretability analysis of the learnt models. We can directly visualise the 2D LSTM hidden states of our oracle experiments, which enables a qualitative comparison of the various model architectures (see Figure 8). Visualising the hidden states with respect to the temporal dependencies indicates that the CSC Instance Space LSTM model has learnt insightful hidden state representations. Breaking down the CSC Instance Space LSTM model hidden embeddings: for the Timer task, time is represented along a curve, with a sparser representation around $t = 50$ (the crossover point when the treasure becomes positive). For the Moving task, time is similarly captured along with an additional notion of the change in treasure direction from right to left. For the Key task, the binary state of no key vs key is separated, with additional partitioning based on x position, denoting the two different start points of the agent. In the Lunar Lander task, the model has learnt a strong separation between states either side of the crossover point when the time on the pad is equal to 50, with high sparsity around the crossover point. In comparison, the Embedding and Instance Space LSTM models have not learnt as sparse a representation. We discuss the Charger task in Section 5.2.

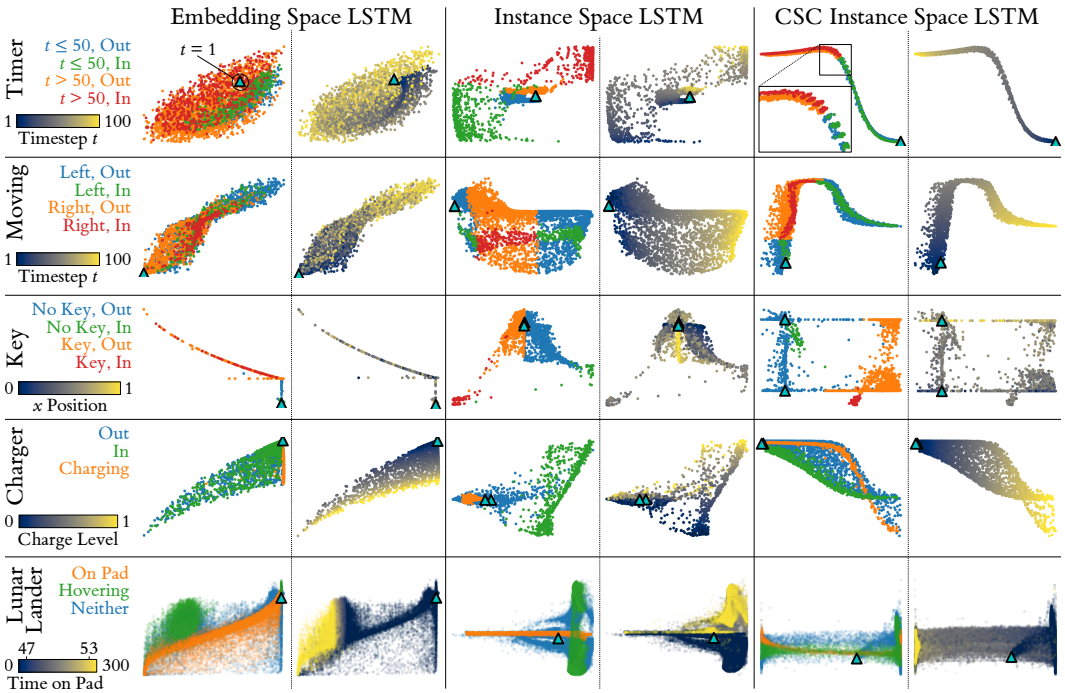

Figure 8: Learnt hidden state embeddings for our MIL RM models. For each model and task, we categorise the hidden state embeddings depending on the true environment state (first column for each model). In and Out environments states indicate whether the agent is in the treasure zone or not, and for the Moving task, Left and Right indicate the direction in which the goal is currently moving. We also provide labelling based on temporal information (second column for each model). Furthermore, we include markers to indicate the hidden states for the centres of the agent spawn zones. In each case, we elected to use the best-performing repeat for each model as assessed by the reward reconstruction (see Table 1). Note, for the Key task, as the temporal information is captured in the state categorisation (No Key vs Key), we use the second column to show the relationship between the hidden embeddings and the agent's x position.

5.2 Trajectory Probing

We further interpret our models by visualising the learnt reward with respect to the environment state, and by using hand-specified *probe* trajectories to verify that the learnt hidden state transitions mimic the true transitions. We present the above for the CSC Instance Space LSTM model on the Charger task in Figure 9 (Appendix G contains similar figures for all other tasks). The top row shows that the model has correctly learnt the relationships between position, charge, and reward (reward increases

in the treasure zone as charge increases). From the probes, we can see how the charge level can be recovered from the hidden states. We also note that the inflection point between under-charging and over-charging is captured, i.e., this is where the optimal charge level lies, subject to some noise based on where the agent starts in the spawn zones. Furthermore, with the Challenging probe, we observe that the learnt hidden states align with the agent moving in and out of the charging zone.

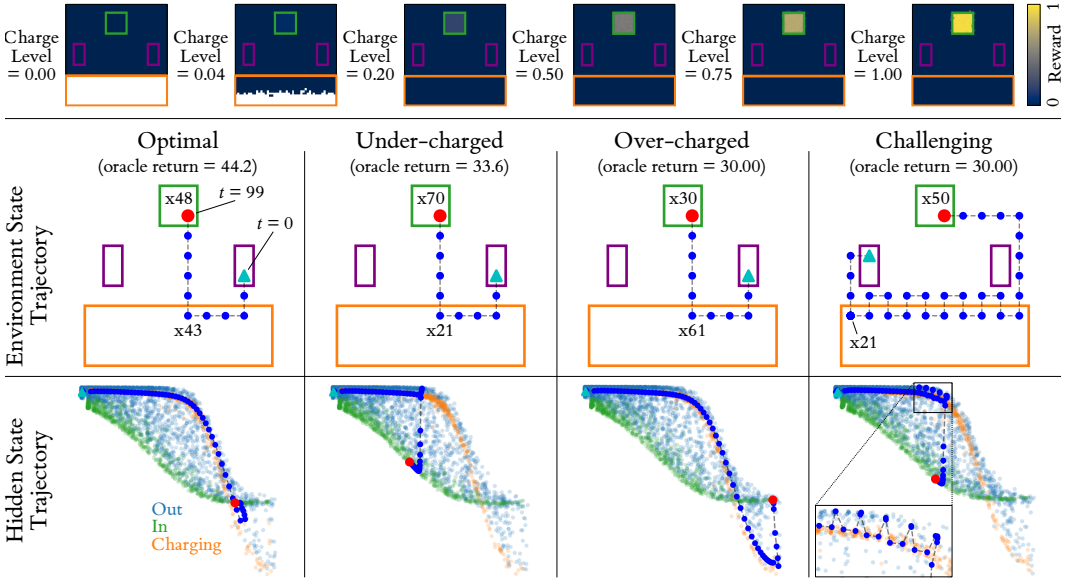

Figure 9: Interpretability for the CSC Instance Space LSTM model on the Charger task. **Top**: the learnt relationship between agent position, charge level, and reward. **Middle/bottom**: Four probe trajectories demonstrating hidden state transitions as the trajectory progresses. *Optimal*: best possible return (charge to sufficient level then maximise time in treasure). *Over-charged*: continuing to charge after maximum charge of 1 is reached. "xn" labels indicate the agent remains in a position for n timesteps. As in Figure 8, we use the best-performing model according to reward reconstruction.

5.3 Limitations and Future Work

Although we analyse the performance of our methods in the presence of noisy labels in Section 4.4, a major area of future work is to apply our methods to human labelling (for a discussion of this, see Appendix C). Another area of future work involves more complex environments, for example the use of tasks with image observations, similar to the Atari environments in Open AI Gym [9]. Furthermore, we perform RM from either an offline dataset or from only one RL training iteration; an iterative bootstrapping approach with multiple RL + RM training iterations could lead to improved RL results. There are also limitations with our MIL RM approach for the Lunar Lander task; see Appendix E.5 for details and suggestions for future work. More generally, we hope that our identification of the link between RM and MIL may inspire a bidirectional transfer of tools and techniques.

6 Conclusion

We posed the problem of non-Markovian RM, which removes an unrealistic assumption about how humans evaluate temporally extended agent behaviours. After identifying an isomorphism between RM and MIL, we proposed and evaluated novel MIL-inspired models that allow us to reconstruct non-Markovian reward functions, augment agent training, and interpret their learnt representations.

Acknowledgements

This work was funded by the AXA Research Fund, the UKRI Trustworthy Autonomous Systems Hub (EP/V00784X/1), and an EPSRC/Thales industrial CASE award. We would also like to thank the Universities of Southampton and Bristol, as well as the Alan Turing Institute, for their support. The authors acknowledge the use of the IRIDIS (Southampton) and BlueCrystal (Bristol) high-performance computing facilities and associated support services in the completion of this work.

References

[1] D. Ariely and Z. Carmon. Gestalt characteristics of experiences: The defining features of summarized events. *Journal of Behavioral Decision Making*, 13(2):191–201, 2000.

[2] F. Bacchus, C. Boutilier, and A. Grove. Rewarding behaviors. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1160–1167, 1996.

[3] F. Bacchus, C. Boutilier, and A. Grove. Structured solution methods for non-markovian decision processes. In *AAAI/IAAI*, pages 112–117. Citeseer, 1997.

[4] A. Bajcsy, D. P. Losey, M. K. O'Malley, and A. D. Dragan. Learning robot objectives from physical human interaction. In *Conference on Robot Learning*, pages 217–226. PMLR, 2017.

[5] B. Bakker. Reinforcement learning with Long Short-term Memory. *Advances in Neural Information Processing Systems*, 14, 2001.

[6] C. Basu, E. Bıyık, Z. He, M. Singhal, and D. Sadigh. Active learning of reward dynamics from hierarchical queries. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 120–127. IEEE, 2019.

[7] K. C. Berridge and J. P. O'Doherty. From experienced utility to decision utility. In *Neuroeconomics*, pages 335–351. Elsevier, 2014.

[8] T. Bewley and F. Lecue. Interpretable Preference-based Reinforcement Learning with Tree-Structured Reward Functions. In *21st International Conference on Autonomous Agents and Multiagent Systems*, 2022.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[10] M.-A. Carbonneau, V. Cheplygina, E. Granger, and G. Gagnon. Multiple instance learning: A survey of problem characteristics and applications. *Pattern Recognition*, 77:329–353, 2018.

[11] L. Chan, A. Critch, and A. Dragan. The impacts of known and unknown demonstrator irrationality on reward inference. 2020.

[12] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30, 2017.

[13] J. Early, C. Evers, and S. Ramchurn. Model Agnostic Interpretability for Multiple Instance Learning. In *International Conference on Learning Representations*, 2022.

[14] S. Griffith, K. Subramanian, J. Scholz, C. L. Isbell, and A. L. Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. *Advances in Neural Information Processing Systems*, 26, 2013.

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[16] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan. Inverse reward design. *Advances in Neural Information Processing Systems*, 30, 2017.

[17] M. Hausknecht and P. Stone. Deep Recurrent Q-learning for Partially Observable MDPs. In *2015 AAAI Fall Symposium Series*, 2015.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[21] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2107–2116. PMLR, 2018.

[22] M. Ilse, J. Tomczak, and M. Welling. Attention-based deep multiple instance learning. In *International conference on machine learning*, pages 2127–2136. PMLR, 2018.

[23] F. Jarboui and V. Perchet. Trajectory representation learning for multi-task nmrdp planning. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 6786–6793. IEEE, 2021.

[24] H. J. Jeon, S. Milli, and A. Dragan. Reward-rational (implicit) choice: A unifying formalism for reward learning. *Advances in Neural Information Processing Systems*, 33:4415–4426, 2020.

[25] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[26] D. Kahneman. Evaluation by moments: Past and future. *Choices, values, and frames*, pages 693–708, 2000.

[27] M. G. Kendall. Rank correlation methods. 1948.

[28] K. Lee, L. Smith, A. Dragan, and P. Abbeel. B-pref: Benchmarking preference-based reinforcement learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[29] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, and S. Legg. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*, 2018.

[30] D. Li and Y. Zhang. Multi-instance learning algorithm based on lstm for chinese painting image classification. *IEEE Access*, 8:179336–179345, 2020.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[32] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, volume 1, page 2. PMLR, 2000.

[33] S. Reddy, A. Dragan, S. Levine, S. Legg, and J. Leike. Learning human objectives by evaluating hypothetical behavior. In *International Conference on Machine Learning*, pages 8020–8029. PMLR, 2020.

[34] D. Rolnick, A. Veit, S. Belongie, and N. Shavit. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017.

[35] S. Russell. *Human Compatible: Artificial Intelligence and the Problem of Control*. Penguin, 2019.

[36] A. Shah, S. Wadhwania, and J. Shah. Interactive robot training for non-markov tasks. *arXiv preprint arXiv:2003.02232*, 2020.

[37] Z. Shao, H. Bian, Y. Chen, Y. Wang, J. Zhang, X. Ji, et al. Transmil: Transformer based correlated multiple instance learning for whole slide image classification. *Advances in Neural Information Processing Systems*, 34, 2021.

[38] K. E. Stanovich. Higher-order preferences and the master rationality motive. *Thinking & Reasoning*, 14(1):111–127, 2008.

[39] S. Thiébaux, C. Gretton, J. Slaney, D. Price, and F. Kabanza. Decision-theoretic planning with non-markovian rewards. *Journal of Artificial Intelligence Research*, 25:17–74, 2006.

[40] J. Tien, J. Z.-Y. He, Z. Erickson, A. D. Dragan, and D. Brown. A study of causal confusion in preference-based reward learning. *arXiv preprint arXiv:2204.06601*, 2022.

[41] R. Toro Icarte, E. Waldie, T. Klassen, R. Valenzano, M. Castro, and S. McIlraith. Learning reward machines for partially observable reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[42] M. Tu, J. Huang, X. He, and B. Zhou. Multiple instance learning with graph neural networks. *arXiv preprint arXiv:1906.04881*, 2019.

[43] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[44] K. Wang, J. Oramas, and T. Tuytelaars. In defense of lstms for addressing multiple instance learning problems. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[45] X. Wang, Y. Yan, P. Tang, X. Bai, and W. Liu. Revisiting multiple instance neural networks. *Pattern Recognition*, 74:15–24, 2018.

[46] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18(5):620–634, 2010.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.3
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Appendix A
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendices D, E, and F
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Appendix A
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix A
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Our source code and models, which are included in the supplementary material and will be made available publicly upon acceptance.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Implementation and Resource Details

This work was implemented in Python 3.8 / 3.10 and the machine learning functionality used PyTorch. All required libraries for our work are given in a `requirements.txt` file. Our code is publicly accessible at https://github.com/JAEarly/MIL-for-Non-Markovian-Reward-Modelling. The majority of MIL model training was carried out on a remote GPU service using a Volta V100 Enterprise Compute GPU with 16GB of VRAM, which utilised CUDA v11.0 to enable GPU support (IRIDIS 5, University of Southampton). For the Lunar Lander task, training each MIL model took a maximum of eight hours. For the other tasks, this was a maximum of two hours. Trained models are included alongside the code. Fixed seeds were used to ensure consistency of dataset splits between training and testing; these are included in the scripts that are used to run the experiments. All our datasets were generated from code; both the scripts to generate the data and also the derived datasets themselves are included alongside our model training code. Dataset generation, as well as all RL agent training, was conducted on a second remote GPU service using a compute node with two Nvidia Pascal P100 cards. Data generation took a maximum of three hours per dataset (BlueCrystal Phase 4, University of Bristol). Agent training for the 2D navigation tasks was computationally light, requiring 8-12 minutes per 400-episode run, although we completed ten repeat runs for each permutation of task and MIL model architecture (one per MIL training repeat). 800-episode RL runs for Lunar Lander took approximately two hours each. Further details on executing the scripts to reproduce our results can be found in the `README.md` file in our code submission.

B Use Cases for Non-Markovian Reward Modelling

The non-Markovian reward formulation applies to cases where rewards depend on hidden state information h_t in addition to environment states s_t and actions a_t, and this information is a function of previous state-action pairs but *not* vice versa (i.e., there is no causal path from h_t to s_{t+k} for any $k > 1$). This crucial caveat distinguishes the formulation from the more general class of partially observable MDPs and demarcates the set of domains to which it can be applied: those involving a secondary Markovian system that "spectates" on events in the environment without intervening. In the RM context, this secondary system is a black box (making its internal state h_t hidden) and explicit rewards are unavailable, being replaced by a sparser and potentially noisy form of reward-dependent feedback (trajectory return labels in our work). Below we identify three classes of use case which fit this technical specification and provide one concrete example for each:

Ambiguous Subtasks Cases involving an extended task with a sequential structure, where it is hard to formally define the conditions for subtask completion, but RM is feasible because a human "knows it when they see it". Here, the hidden state to be learnt represents the current subtask and any auxiliary information needed to determine its completion status.

- **Concrete Example:** Using judges' scores to learn a performative display (e.g., gymnastics, aerobatics) chaining several manoeuvres whose start and end conditions are difficult to formalise *a priori*. This could be considered as an extension of the single backflip task studied in the foundational RM work by Christiano et al. [12].

Dependencies on Subjective Affect Cases where a human's reward function is dependent on their affective (emotional) status, which in turn depends on their prior experiences. Assuming this information is not directly available in the observed environment state, it must be inferred from data.

- **Concrete Example:** Using periodic satisfaction ratings to train a personal assistant robot whose owner's mood, needs and preferences vary from day to day. These variations may influence the preferred driving style of a chauffeur service or choice of evening meal.

Irrationalities/Cognitive Biases Cases where one or more forms of bias colour a human's post hoc rating of an observed trajectory, even if their instantaneously-experienced reward is Markovian. Psychological studies of how humans aggregate immediate rewards into retrospective evaluations of the quality of an experience find that a straight summation assumption is unrealistic, with subjects exhibiting high sensitivity to contrast effects and recency bias (collectively termed the peak-end rule)

[26], and factoring in anticipated future states in addition to those actually observed [1]. Here, the hidden state captures an aggregate representation of the biases at play in a given human's evaluation.[5]

- **Concrete Example:** Using customer "star ratings" to improve a holiday planning agent whose recommendations aim to account for an unknown mix of biases such as the peak-end rule. The agent may use the learnt bias model to prioritise key moments in a holiday when managing the travel schedule and budget, in order to maximise future customers' star ratings.

Finally, we note that it is a matter of taste as to whether hidden state information is framed as situated *inside a human evaluator's mind* or *in the environment but only visible to the human*. It is not technically necessary to decide between these two framings, as the mathematical problem of non-Markovian RM is equivalent. From an agent's perspective, a human evaluator is part of an augmented environment, even if they never intervene directly to influence the state.

C Comparing Human and Oracle Labelling

Oracle-based experiments are often used to evaluate RM methods since they enable scalable quantitative validation [14, 16, 33]. However, we identify three concrete differences between our oracle preference labelling method and realistic human labelling: 1) preference form, 2) preference sparsity, and 3) preference noise. Preference form is the different ways of providing labels; in this work we used return values which are highly informative and easy to learn from, but it has long been understood that humans find it easier to give less direct feedback, such as pairwise rankings [12] or good/bad/neural labels [27]. Preference sparsity occurs when the time- and cost-expensiveness of eliciting human labels reduces the proportion of data that can be labelled, and preference noise arises from uncertainties in the human labelling process (as opposed to perfect oracle labelling). We decided to focus on noise in this work as it is an established way of making oracle experiments more realistic [28], and also fits in with our discussion of human uncertainties and cognitive biases. Our experiments in Section 4.4 indicate that our methods degrade gracefully in the presence of noise, which gives us some confidence that they will transfer well to human labels. However, future work should consider preference sparsity and form, the latter of which will involve modifying the data collection pipeline and loss function (e.g., to a contrastive loss in the case of pairwise rankings). Beyond accounting for these three differences whilst still using oracle labels, the next step would be to conduct evaluations using actual human labels.

D Preliminary Experiments on Toy Datasets

In this section, we give detail our preliminary experiments that were run on toy problems to initially develop and validate our approach. Below we outline the datasets (Section D.1), models and training hyperparameters (Section D.2), and results (Section D.3) of these experiments.

D.1 Datasets

We introduce three toy datasets, each abstracted from the RL context, to act as benchmark tests for our models. Each of these datasets uses ordered bags comprised of two-dimensional instances, where each instance has an associated label (called "reward" below, for consistency), and the overall bag label (return) is the sum of the instance labels.

Toggle Switch An instance is the position of a toggle switch ts and a value v; if the switch is on ($ts = 1$), then the reward for the instance is v; otherwise the reward for the instance is 0. Here, there

[5] A fascinating philosophical question arises here. When (following the method of Section 3.3) an RL agent is trained using a non-Markovian RM model that captures a cognitive bias, should the agent learn to maximise rewards *including* the bias (which, for example, might lead it to prioritise its peak and final reward rather than seek uniformly good performance), or *exluding* it (which would revert to uniform prioritisation). This issue of whether intelligent agents should seek to exploit human irrationalities when optimising for their revealed preferences, or appeal to their unbiased "better angels", relates to distinctions between first- and second-order preferences [38] or between experienced and remembered (or decision) utility [7]. We defer this question to those with more relevant expertise but note that regardless of the answer, it is essential to include the biases in the reward model using a method such as the one proposed in this work.

is no hidden information, as it is possible to calculate the reward for an instance from its contents ts, v alone. This serves as a null example to elucidate what happens in a Markovian setting.

Push Switch We modify the toggle switch setup so that the instance now represents a push switch ($p = 1$ if this switch is pressed), where pressing the switch flips a binary hidden state ts (the same information as was previously represented by the toggle switch). This hidden state then determines the reward as before (v if $ts = 0$, else 0). As ts must be tracked between successive instances, it is not possible to determine the reward for an instance solely by observing its contents p, v, so this setup is non-Markovian.

Dial We generalise the hidden state from a binary switch to a continuous-valued dial. Given an instance m, v, the dial's current value d is moved up or down by m. The reward is then given as $d \cdot v$ The problem remains non-Markovian, but now the hidden state that needs tracking, d, is continuous rather than discrete.

D.2 Models and Hyperparameters

When training the MIL models on the toy dataset, we used the Adam optimiser with a batch size of one (i.e., one bag per batch) to minimise mean squared error (MSE) loss. Training was performed using validation loss early stopping, i.e., if the validation loss did not decrease after a certain number of training epochs (patience value), we terminated the training and selected the model at which the validation loss was lowest. If the patience value was not reached (i.e., the validation loss kept decreasing), we terminated training after a maximum number of epochs had been reached, and again selected the model at which the validation loss was lowest. The hyperparameters for training the models on each dataset (including learning rate (LR) and weight decay (WD)) are given in Table A1. Dropout was not used. These hyperparameters were found through a small amount of trial and error, i.e., no formal hyperparameter tuning was carried out.

Table A1: Toy dataset MIL training hyperparameters.

Dataset	LR	WD	Patience	Epochs
Toggle Switch	1×10^{-4}	1×10^{-5}	20	100
Push Switch	1×10^{-3}	0	30	150
Dial	1×10^{-3}	0	30	150

In Tables A2 to A5 we give the architectures for the MIL models we used in the toy dataset experiments. The models are a combination of fully connected layers (FC) along with different MIL pooling mechanisms. Rectified linear unit (ReLU) activation is applied to the FC hidden layers. We label the layers based on the part of the network they belong to: feature extractor (FE), head network (HN), or pooling (P); see Section 3.2. We also indicate the input and output sizes: b x n indicates an input or output where there is a representation of length n for each of the b instances (b is the size of the input bag).

Table A2: Toy Instance Space NN

Layer	Type	Input	Output
1 (FE)	FC + ReLU	b x 2	b x 2
2 (HN)	FC	b x 2	b x 1
3 (P)	mil-sum	b x 1	1

Table A3: Toy Embedding Space LSTM

Layer	Type	Input	Output
1 (FE)	FC + ReLU	b x 2	b x 2
2 (P)	mil-emb-lstm	b x 2	2
3 (HN)	FC	2	1

D.3 Results

For each of the toy datasets, we generate 5000 random bags with between 10 and 20 instances per bag (uniformly distributed). We use an 80/10/10 dataset split for training, validation, and testing, and repeat our experiments with ten different variations of this split (so in total we have ten repeats of each model type for each dataset). We show results for both return and reward reconstruction for the toy datasets in Table A6. From these results, we can make several observations. Firstly, as

Table A4: Toy Instance Space LSTM

Layer	Type	Input	Output
1 (FE)	FC + ReLU	b x 2	b x 2
2 (P-1)	mil-ins-lstm	b x 2	b x 2
3 (HN)	FC	b x 2	b x 1
4 (P-2)	mil-sum	b x 1	1

Table A5: Toy CSC Instance Space LSTM

Layer	Type	Input	Output
1 (FE)	FC + ReLU	b x 2	b x 2
2 (P-1)	mil-csc-ins-lstm	b x 2	b x 2
3 (HN)	FC	b x 2	b x 1
4 (P-2)	mil-sum	b x 1	1

expected, the Instance Space NN architecture only works on the Markovian Toggle Switch dataset, i.e., it fails on the non-Markovian Push Switch and Dial datasets as it is unable to deal with temporal dependencies. We also note that our two proposed architectures (Instance Space LSTM and CSC Instance Space LSTM) outperform the baseline Embedding Space LSTM method on both return and reward, with the CSC Instance Space LSTM model providing the best results overall. Finally, we observe that a better return performance does not always guarantee better reward performance: for the Dial dataset, the Instance Space LSTM makes better return predictions than the CSC Instance Space LSTM model, but worse reward predictions. A similar outcome can be seen for the Embedding Space LSTM and the Instance Space NN on the Push Switch dataset.

Table A6: Toy dataset return (top) and reward (bottom) results. Each measurement is the mean MSE averaged over ten repeats, with the standard errors of the mean also given. Bold entries indicate the best-performing model for each (metric, dataset) pair.

Model	Toggle Switch	Push Switch	Dial	Overall
Instance Space NN	0.030 ± 0.029	3.337 ± 0.054	5.489 ± 0.157	2.952
Embedding Space LSTM	0.008 ± 0.002	0.663 ± 0.194	0.434 ± 0.075	0.368
Instance Space LSTM	0.062 ± 0.058	0.262 ± 0.154	$\mathbf{0.111 \pm 0.014}$	0.145
CSC Instance Space LSTM	$\mathbf{0.000 \pm 0.000}$	$\mathbf{0.140 \pm 0.065}$	0.121 ± 0.043	**0.087**
Instance Space NN	0.002 ± 0.002	0.086 ± 0.001	0.954 ± 0.011	0.347
Embedding Space LSTM	0.003 ± 0.001	0.206 ± 0.100	0.244 ± 0.077	0.151
Instance Space LSTM	0.004 ± 0.004	0.021 ± 0.008	0.026 ± 0.004	0.017
CSC Instance Space LSTM	$\mathbf{0.000 \pm 0.000}$	$\mathbf{0.012 \pm 0.004}$	$\mathbf{0.022 \pm 0.007}$	**0.011**

E RL Task Details, Data Generation, and Model Hyperparameters

In this section, we give more detail on the reward reconstruction experiments for the RL tasks. First, we give more information about the RL tasks (Section E.1), then explain how we generated datasets from the tasks (Section E.2), and give the MIL model architectures and hyperparameters used in the Timer, Moving, Key and Charger tasks (Section E.3), and the Lunar Lander task (Section E.4). Finally, we discuss the limitations of our approach to using MIL RM for the Lunar Lander task (Section E.5).

E.1 Task Details

The first four tasks are implemented in Python within a common 2D simulator following the OpenAI Gym standard [9]. The agent's position x, y is moved by one of five discrete actions: up, down, left, right and no-op. In the first four cases, the position is moved by 0.1 in the specified direction. The motion vector is then corrupted by zero-mean Gaussian noise with a standard deviation of 0.02 in both x and y and clipped into the bounds $[0, 1]^2$. Zones of interest (spawn zones, treasure, key, charger) are specified as rectangles lying within these bounds. At time t, the environment state s_t (which is directly observed by the MIL RM models) is the 2D vector of the current position $[x_t, y_t]$; its dynamics are Markovian given the agent's chosen action. The hidden state h_t is the task-specific information that renders the oracle's reward function R Markovian:

- **Timer:** $h_0 = 0$ and $h_{t+1} = \delta(h_t, s_t, a_t) = h_t + 1$; the hidden state simply tracks the current timestep index. Reward is given by[6]

$$R(s_t, a_t, h_{t+1}) = \text{in_treasure}(s_t) \cdot \begin{cases} -1 & \text{if } h_{t+1} \leq 50, \\ +1 & \text{otherwise,} \end{cases}$$

  where $\text{in_treasure}([x_t, y_t]) = 1$ if $0.4 \leq x_t \leq 0.6$ and $0.7 \leq y_t \leq 0.9$, and 0 otherwise.

- **Moving:** $h_0 = [0.4, -0.02]$, the initial horizontal position (left edge) and velocity of the moving treasure rectangle. Hidden state dynamics encode the left-right oscillation:

$$h_{t+1} = \delta(h_t, s_t, a_t) = \left[h_t^0 + h_t^1, \begin{cases} h_t^1 & \text{if } 0 < (h_t^0 + h_t^1) < 0.8, \\ -h_t^1 & \text{otherwise} \end{cases} \right].$$

  Reward is given by $R([x_t, y_t], a_t, h_{t+1}) = 1$ if $h_{t+1} \leq x_t \leq (h_{t+1} + 0.2)$ and $0.7 \leq y_t \leq 0.9$, and 0 otherwise.

- **Key:** $h_0 = 0$, indicating that the agent initialises without the key. The key collection dynamics are encoded by

$$h_{t+1} = \delta(h_t, [x_t, y_t], a_t) = \begin{cases} 1 & \text{if } 0.4 \leq x_t \leq 0.6 \text{ and } 0.1 \leq y_t \leq 0.3, \\ h_t & \text{otherwise.} \end{cases}$$

  Reward is given by $R(s_t, a_t, h_{t+1}) = \text{in_treasure}(s_t) \cdot h_{t+1}$, where the in_treasure function is the same as in the Timer task.

- **Charger:** $h_0 = 0$, indicating an initial charge level of zero. The charging dynamics are encoded by

$$h_{t+1} = \delta(h_t, [x_t, y_t], a_t) = \begin{cases} \min(h_t + 0.02, 1) & \text{if } y_t \leq 0.3, \\ h_t & \text{otherwise.} \end{cases}$$

  Reward is given identically to the Key task, $R(s_t, a_t, h_{t+1}) = \text{in_treasure}(s_t) \cdot h_{t+1}$.

The **Lunar Lander** task is a modified version of the LUNARLANDERCONTINUOUS-V2 baseline included as standard in the OpenAI Gym library [9]. We leave the state and action spaces unmodified. The 8D state vector is $[x, y, v^x, v^y, \theta, \dot{\theta}, c^l, c^r]$, where x, y and v^x, v^y are the landing craft's horizontal and vertical positions and velocities, θ and $\dot{\theta}$ are its angle from vertical and angular velocity, and c^l, c^r are two binary contact detectors indicating whether the left and right landing legs are in contact with the ground. The 2D continuous action $[u^m, u^s]$ is a pair of throttle values for two engines: main u^m and side u^s. We also retain the default initialisation conditions (the lander spawns in a narrow zone above the landing pad, with slightly-randomised orientation and velocities), the automatic termination of episodes when $|x|$ exceeds 1 (i.e., when the lander leaves the rendered screen area), and the physics that determine how the lander responds to engine activations. However, we replace the standard reward function with an oracle that rewards the agent for landing on the pad for up to 50 timesteps, and then taking off again to hover within a target zone until an episode time limit ($T = 500$) is reached. Rendering this two-stage objective Markovian requires a hidden state h_t that tracks the number of timesteps spent on the pad so far. Formally, reward is given by

$$R(s_t, a_t, h_{t+1}) = \begin{cases} R_{\text{pad}}(s_t) + R_{\text{shaping}}(s_t, 0) & \text{if } h_{t+1} < 50, \\ R_{\text{no_contact}}(s_t) + R_{\text{hover}}(s_t) + R_{\text{shaping}}(s_t, 1) & \text{otherwise,} \end{cases}$$

where R_{pad} rewards the agent for being central with both legs on the ground (i.e., on the pad),

$$R_{\text{pad}}([x_t, y_t, v_t^x, v_t^y, \theta_t, \dot{\theta}_t, c_t^l, c_t^r]) = \begin{cases} 1 & \text{if } -0.2 \leq x_t \leq 0.2 \text{ and } c_t^l = 1 \text{ and } c_t^r = 1, \\ 0 & \text{otherwise,} \end{cases}$$

$R_{\text{no_contact}}$ rewards breaking leg-ground contact,

$$R_{\text{no_contact}}([x_t, y_t, v_t^x, v_t^y, \theta_t, \dot{\theta}_t, c_t^l, c_t^r]) = \begin{cases} 1 & \text{if } c_t^l = 0 \text{ and } c_t^r = 0, \\ 0 & \text{otherwise,} \end{cases}$$

[6] Note the timestep indices used here, which result from the order in which environment states, hidden states and rewards are computed. At time t, the hidden state h_t is first updated to h_{t+1} by $\delta(h_t, s_t, a_t)$, then the reward is computed as $R(s_t, a_t, h_{t+1})$, and finally the environment state is updated to s_{t+1} by $D(s_t, a_t)$.

R_{hover} rewards aerial positions in a target zone above the pad,

$$R_{\text{hover}}([x_t, y_t, v_t^x, v_t^y, \theta_t, \dot{\theta}_t, c_t^l, c_t^r]) = \begin{cases} 1 & \text{if } -0.5 \leq x_t \leq 0.5 \text{ and } 0.75 \leq y_t \leq 1.25, \\ 0 & \text{otherwise,} \end{cases}$$

and R_{shaping} promotes slow, stable, central flight towards a target vertical position y_{target},

$$R_{\text{shaping}}([x_t, y_t, v_t^x, v_t^y, \theta_t, \dot{\theta}_t, c_t^l, c_t^r], y_{\text{target}}) = \\ 0.1 \times \max\left(2 - \left(\sqrt{(x_t)^2 + (y_t - y_{\text{target}})^2} + \sqrt{(v_t^x)^2 + (v_t^y)^2} + |\theta_t| + |\dot{\theta}_t|\right),\ 0\right).$$

The hidden state dynamics are

$$h_{t+1} = \begin{cases} \min(h_t + 1, 50) & \text{if } R_{\text{pad}}(s_t) = 1, \\ h_t & \text{otherwise,} \end{cases}$$

with $h_0 = 0$.

E.2 MIL Dataset Generation

We obtain datasets of several thousand trajectories per task, containing a wide distribution of outcomes and return values, as follows. For each task k, we define a discrete *trajectory classifier* function $C_k : \Xi \to \mathcal{C}_k$ and a limit p_k on the proportion of trajectories in the dataset that are allowed to map to each class in $\mathcal{C}_k$. These are given as follows:

- **Timer:** $\mathcal{C}_{\text{timer}} = \text{num_neg} \times \text{num_pos}$, where $\text{num_neg} = \{0..50\}$ counts the number of timesteps the agent spends in the treasure while its reward is negative ($t \leq 50$), and $\text{num_pos} = \{0..50\}$ counts the number while the reward is positive ($t > 50$). The number of classes is $|\mathcal{C}_{\text{timer}}| = 51^2 = 2601$ and the per-class limit is $p_{\text{timer}} = 0.002$.
- **Moving:** $\mathcal{C}_{\text{moving}} = \text{num_treasure}$, where $\text{num_treasure} = \{0..100\}$ counts the timesteps spent in the treasure. $|\mathcal{C}_{\text{moving}}| = 101$ and $p_{\text{moving}} = 0.05$.
- **Key:** $\mathcal{C}_{\text{key}} = \{\text{no_key}, \text{key_no_treasure}, \text{treasure}\}$, where the class is no_key if the key is not collected, key_no_treasure if the key is collected but the treasure is not reached, and treasure if the treasure is reached after collecting the key. $|\mathcal{C}_{\text{key}}| = 3$ and p_{key} is defined on a per-class basis: 0.25 for no_key and key, and 0.5 for treasure.
- **Charger:** $\mathcal{C}_{\text{charger}} = \text{num_treasure} \times \text{charge_bin}$, where $\text{num_treasure} = \{0..100\}$ counts the timesteps spent in the treasure and $\text{charge_bin} = \{1..20\}$ is a binned representation of the *mean* charge level when in the treasure (e.g., 0.0 maps to bin 1, 0.48 to bin 10, 0.96 to bin 20). $|\mathcal{C}_{\text{charger}}| = 2020$ and $p_{\text{charger}} = 0.002$.
- **Lunar Lander:** $\mathcal{C}_{\text{lunar}} = \text{pad_bin} \times \text{take_off} \times \text{hover_bin}$, where $\text{pad_bin} = \{0, \{1..49\}, 50+\}$ is a binned representation of the number of timesteps spent on the landing pad (i.e., zero, fewer than 50 or at least 50), $\text{take_off} = \{0, 1\}$ is a binary indicator of whether the lander takes off again after being on the pad, and $\text{hover_bin} = \{0, \{1..19\}, 20+\}$ is a binned representation of the number of timesteps spent in the hover zone after being on the pad.[7] $|\mathcal{C}_{\text{lunar}}| = 18$, of which 9 are actually realisable (e.g., the lander cannot take off from the pad if it never reached it in the first place) and $p_{\text{lunar}} = 0.2$.

For $k \in \{\text{timer}, \text{moving}, \text{key}, \text{charger}\}$, a dataset $\mathbf{X}_k$, is assembled iteratively. On each iteration, we generate a length-100 trajectory, ξ, by sampling agent actions uniform-randomly from the action space (up, down, left, right, no-op) and running them through the simulator. Once the trajectory is complete, we evaluate its class $C_k(\xi)$. If there are already at least $p_k \times 5000$ trajectories in $\mathbf{X}_k$ with this class, ξ is discarded. Otherwise, it is added to $\mathbf{X}_k$. This process repeats until $|\mathbf{X}_k| = 5000$.

The state-action space for Lunar Lander is too large for a random generate-and-select algorithm to terminate in any reasonable time. Instead, we recycle the length-500 trajectories generated as a by-product of training the oracle-based RL baselines (black curves in Figure 5, plus six more runs not included in the figure). Starting from a bank of 12000 trajectories, filtering based on the

[7] If the lander reaches the target of 50 timesteps on the pad, the time in the hover zone is measured from this point onwards. Otherwise, it is measured from the first timestep that the lander leaves the pad.

threshold $p_{\text{lunar}} = 0.2$ yields a final dataset $\mathbf{X}_{\text{lunar}}$ with 9762 trajectories. Although this approach of relying on oracle-trained agents to generate data may initially appear to "put the cart before the horse", we suggest that it provides a valuable test of the ability of our MIL models to learn from goal-directed (c.f. random) trajectories, and thus is a step closer to the online bootstrapping approach of simultaneous RM and RL, which we aim to tackle in future work (see Section 5.3).

E.3 MIL Models and Hyperparameters

The MIL model training on the Timer, Moving, Key, and Charger tasks used the same process as for the toy model training (see Section D.2). However, we also applied dropout (DO) in these models. We give the MIL training hyperparameters for each of these tasks in Table A12. Again, the hyperparameters were found through a small amount of trial and error, i.e., no formal hyperparameter tuning was carried out.

Table A7: Timer, Moving, Key, and Charger task training hyperparameters.

Dataset	LR	WD	DO	Patience	Epochs
Timer	5×10^{-4}	0	0.1	50	250
Moving	5×10^{-4}	0	0.1	50	250
Key	5×10^{-4}	0	0.1	30	150
Charger	5×10^{-4}	0	0.1	50	250

In Tables A8 to A11 we give the architectures for the MIL models we used in the Timer, Moving, Key, and Charger tasks. As in the toy dataset experiments, the models are a combination of fully connected layers (FC) along with different MIL pooling mechanisms. Rectified linear unit (ReLU) activation is applied to the FC hidden layers. Again, we label the layers based on the part of the network they belong to: feature extractor (FE), head network (HN), or pooling (P); see Section 3.2. We also indicate the input and output sizes: b x n indicates an input or output where there is a representation of length n for each of the b instances (b is the size of the input bag). Input features were normalised using mean/standard deviation scaling.

Table A8: RL Instance Space NN

Layer	Type	Input	Output
1 (FE-1)	FC + ReLU + DO	b x 2	b x 64
2 (FE-2)	FC + ReLU + DO	b x 64	b x 32
3 (FE-3)	FC + ReLU + DO	b x 32	b x 32
4 (HN-1)	FC + ReLU + DO	b x 32	b x 32
5 (HN-2)	FC + ReLU + DO	b x 32	b x 16
6 (HN-3)	FC	b x 16	b x 1
7 (P)	mil-sum	b x 1	1

Table A9: RL Embedding Space LSTM

Layer	Type	Input	Output
1 (FE-1)	FC + ReLU + DO	b x 2	b x 64
2 (FE-2)	FC + ReLU + DO	b x 64	b x 32
3 (FE-3)	FC + ReLU + DO	b x 32	b x 32
4 (P)	mil-emb-lstm	b x 32	2
5 (HN-1)	FC + ReLU + DO	2	32
6 (HN-2)	FC + ReLU + DO	32	16
7 (HN-3)	FC	16	1

Table A10: RL Instance Space LSTM

Layer	Type	Input	Output
1 (FE-1)	FC + ReLU + DO	b x 2	b x 64
2 (FE-2)	FC + ReLU + DO	b x 64	b x 32
3 (FE-3)	FC + ReLU + DO	b x 32	b x 32
4 (P-1)	mil-ins-lstm	b x 32	b x 2
5 (HN-1)	FC + ReLU + DO	b x 2	b x 32
6 (HN-2)	FC + ReLU + DO	b x 32	b x 16
7 (HN-3)	FC	b x 16	b x 1
8 (P-2)	mil-sum	b x 1	1

Table A11: RL CSC Instance Space LSTM

Layer	Type	Input	Output
1 (FE-1)	FC + ReLU + DO	b x 2	b x 64
2 (FE-2)	FC + ReLU + DO	b x 64	b x 32
3 (FE-3)	FC + ReLU + DO	b x 32	b x 32
4 (P-1)	mil-csc-ins-lstm	b x 32	b x 2
5 (HN-1)	FC + ReLU + DO	b x 34	b x 32
6 (HN-2)	FC + ReLU + DO	b x 32	b x 16
7 (HN-3)	FC	b x 16	b x 1
8 (P-2)	mil-sum	b x 1	1

E.4 Lunar Lander MIL Models and Hyperparameters

There are several differences between the MIL training process for the Lunar Lander task and the other four RL tasks (see Appendix E.3). Firstly, the reward targets (and as such, return targets) were scaled down by a factor of 100 in order to avoid extremely large gradients from high prediction targets. For example, a trajectory with an original return of 700 would have a scaled return of 7. Secondly, the input data was scaled linearly between -0.5 and 0.5 (using the minimum and maximum range of each feature). This was found to give more consistent feature ranges than mean/standard deviation scaling as was used in the other tasks (this was due to large outliers in certain features, e.g., the rotational features were largely clustered around 0, but had extreme values up to $\pm$ 90). We give the training hyperparameters for the lunar lander environment in Table A12. Again, no formal hyperparameter tuning was carried out, so better performance of these models could potentially be achieved with better parameters (including shorter training times with a higher learning rate).

Table A12: Lunar Lander MIL training hyperparameters.

Dataset	LR	WD	DO	Patience	Epochs
Lunar Lander	1×10^{-4}	0	0	30	200

We used similar architectures to the other four RL tasks (see Appendix E.3), but with larger layers; see Tables A13 through A16. However, the depth of the models remained the same, as did the size of the hidden state embedding. Also note the addition of the Leaky ReLU activation function in the head networks, which we discuss further in Appendix E.5.

Table A13: Lunar Lander Instance Space NN

Layer	Type	Input	Output
1 (FE-1)	FC + ReLU + DO	b x 2	b x 128
2 (FE-2)	FC + ReLU + DO	b x 128	b x 64
3 (FE-3)	FC + ReLU + DO	b x 64	b x 64
4 (HN-1)	FC + ReLU + DO	b x 64	b x 64
5 (HN-2)	FC + ReLU + DO	b x 64	b x 32
6 (HN-3)	FC + Leaky ReLU	b x 32	b x 1
7 (P)	mil-sum	b x 1	1

Table A14: Lunar Lander Emb. Space LSTM

Layer	Type	Input	Output
1 (FE-1)	FC + ReLU + DO	b x 2	b x 128
2 (FE-2)	FC + ReLU + DO	b x 128	b x 64
3 (FE-3)	FC + ReLU + DO	b x 64	b x 64
4 (P)	mil-emb-lstm	b x 64	2
5 (HN-1)	FC + ReLU + DO	2	64
6 (HN-2)	FC + ReLU + DO	64	32
7 (HN-3)	FC + Leaky ReLU	32	1

Table A15: Lunar Lander Ins. Space LSTM

Layer	Type	Input	Output
1 (FE-1)	FC + ReLU + DO	b x 2	b x 128
2 (FE-2)	FC + ReLU + DO	b x 128	b x 64
3 (FE-3)	FC + ReLU + DO	b x 64	b x 64
4 (P-1)	mil-ins-lstm	b x 64	b x 2
5 (HN-1)	FC + ReLU + DO	b x 2	b x 64
6 (HN-2)	FC + ReLU + DO	b x 64	b x 32
7 (HN-3)	FC + Leaky ReLU	b x 32	b x 1
8 (P-2)	mil-sum	b x 1	1

Table A16: Lunar Lander CSC Ins. Space LSTM

Layer	Type	Input	Output
1 (FE-1)	FC + ReLU + DO	b x 2	b x 128
2 (FE-2)	FC + ReLU + DO	b x 128	b x 64
3 (FE-3)	FC + ReLU + DO	b x 64	b x 64
4 (P-1)	mil-csc-ins-lstm	b x 64	b x 2
5 (HN-1)	FC + ReLU + DO	b x 66	b x 64
6 (HN-2)	FC + ReLU + DO	b x 64	b x 32
7 (HN-3)	FC + Leaky ReLU	b x 32	b x 1
8 (P-2)	mil-sum	b x 1	1

E.5 Lunar Lander MIL Discussion

In Table 1, we present results for Lunar Lander using only the top five performing models by reward MSE (50% of all models). In this section, we discuss why this is the case.

When training the Lunar Lander architectures with linear activation functions in the head networks, we found that the models were struggling to learn the correct return predictions as they were making negative reward predictions. We know *a priori* that the Lunar Lander task only has positive rewards, therefore we added a Leaky ReLU activation (with a negative slope of 1×10^{-6}) to each architecture's head network to encourage positive predictions. This led to an immediate improvement

in performance for all model architecture (excluding the Instance Space NN baseline, but this was expected to fail on this task as it cannot model temporal dependencies).

However, we found that a proportion of model initialisations remained unable to overcome a certain local minimum during training (corresponding to a return prediction MSE of around 2.1). In other cases, after this threshold was passed, the model performance would rapidly improve. Note this problem occurred in each of the LSTM-based models (the Instance Space NN architecture was never observed to pass this threshold). Therefore, we focus our evaluation on the top 50% of models, which is equivalent to discarding the worse-performing models, the majority of which had not passed the problematic threshold during training.

We investigate this training issue further in Figure A1, focusing specifically on the CSC Instance Space LSTM models. Although the Leaky ReLU activation encourages models to make positive predictions, we can see that it does not prevent them entirely. Furthermore, the models that are not able to cross the return threshold of 2.1 tend to output a greater proportion of negative reward predictions. We thus hypothesise that a better mechanism for preventing negative reward prediction would increase the chance that a given model training run achieves a return loss of less than 2.1. Below we list several such mechanisms as alternatives to our current Leaky ReLU approach.

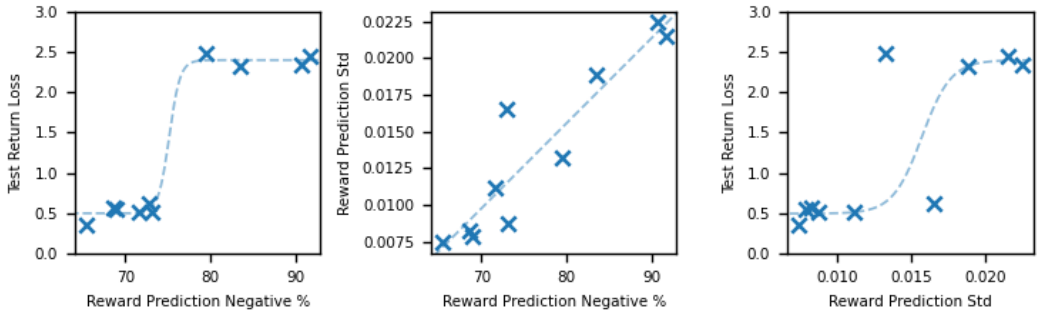

Figure A1: An analysis of the negative reward prediction of the ten Lunar Lander CSC Instance Space LSTM models trained in this work. Note we observed similar trends for the Embedding Space LSTM and Instance Space LSTM models. **Left:** Models that are able to cross the return loss threshold of 2.1 make fewer negative reward predictions (given here as a percentage of all reward predictions) than those that cannot. **Middle:** As the number of negative reward predictions increases, so too does the standard deviation of the reward predictions. In order to sum to the correct return predictions, the positive reward predictions must increase to compensate for the negative reward predictions, leading to a larger variance, and ultimately worse reward prediction. **Right:** The standard deviation of the reward predictions also highlights the loss threshold of 2.1, although not as clearly as the percentage of negative reward predictions.

Replace Leaky ReLU with ReLU One option to remove negative reward predictions entirely is to use ReLU rather than Leaky ReLU in the final activation function of the head networks. However, in the case that all the reward predictions are negative, the gradient of the network will be zero, so no learning can take place. The ability of the network to learn is entirely dependent on achieving at least one positive prediction in order to generate non-zero gradients, which is determined by the initial network weights. This could potentially be overcome with different network initialisation approaches, or by simply discarding training runs that fail to begin to learn.

Replace Leaky ReLU with Sigmoid Instead of using a Leaky ReLU or ReLU activation in the head network, the Sigmoid activation function could be used. This would overcome the gradient issue presented with the use of ReLU, and would ensure that no negative rewards are predicted. However, this would require *a priori* knowledge of the maximum reward target in order to scale the network outputs correctly, and the non-linear activation could make accurate prediction of rewards more difficult in some cases.

Remove target normalisation A further approach to reduce the number of negative reward predictions would be to remove, or at least reduce, the reward target scaling. As discussed above, this was initially included to avoid very large gradients. However, reducing this scaling would move the

average reward prediction away from zero, potentially reducing the number of negative predictions. Care would have to be taken to not reintroduce the large gradient problem.

Regularise reward prediction variance A final alternative approach could be to introduce an additional loss term that encourages the reward predictions to have low variance. This would deter a large range of reward predictions, leading indirectly to fewer negative reward predictions (see Figure A1). However, this approach is only applicable to the LSTM models that produce instance predictions, i.e., it cannot be used with the Embedding Space LSTM network as that architecture does not produce reward predictions during training. Furthermore, this would result in an additional hyperparameter that requires tuning (a coefficient for the new loss term).

F Further Details on RL Agent Training

Adopting OpenAI Gym terminology [9], non-Markovian reward functions (both ground truth oracles and learnt LSTM-based models) are implemented as *wrappers* on rewardless base environments. The role of a wrapper is to track the hidden state of either the oracle or the LSTM throughout an episode and use this to compute rewards to return to the agent. In the "Oracle (without hidden state)" baseline, we return the raw environment state (e.g., the 2D position $[x_t, y_t]$) to the agent unmodified. Otherwise, we concatenate the post-update hidden state h_{t+1} onto the end of the environment state, thereby expanding the state space from the agent's perspective and making rewards Markovian.

This wrapper-based approach allows us to use a completely vanilla RL algorithm. For the 2D navigation tasks, we use a Deep Q-Network (DQN) agent [31] with the double Q-learning trick [43] enabled. For Lunar Lander, which has a continuous action space, we use Soft Actor-Critic (SAC) [15]. In both cases we use a value network with ReLU activation functions, which is updated on every timestep by sampling batches of size 128 from a replay buffer. Bellman updates use a discount factor of $\gamma = 0.99$ and are implemented by the Adam optimiser with a learning rate of 1e^{-3}. A target network tracks the primary one by Polyak averaging of parameters with a coefficient of 5e^{-3} per timestep. Additional hyperparameters are given below:

- **2D tasks (DQN):** number of training episodes $= 400$; replay buffer capacity $= 5\text{e}^4$; value network hidden layer sizes $= [256, 128, 64]$; policy greediness ϵ linearly decayed from 1 to 0.05 over the first 200 episodes and held constant thereafter.
- **Lunar Lander (SAC):** number of training episodes $= 800$; replay buffer capacity $= 1\text{e}^5$; value/policy network hidden layer sizes $= [256, 256]$; policy entropy regularisation coefficient $\alpha = 0.2$; policy updates with Adam optimiser (learning rate 1e^{-4}).

G Trajectory Probing for Other RL Tasks

In this section, we provide additional trajectory probing plots (like Figure 9) for the Timer (Figure A2), Moving (Figure A3), Key (Figure A4), and Lunar Lander (Figure A5) tasks. As before, in these probing plots, we analyse the best-performing CSC Instance Space LSTM model for each task (according to the reward reconstruction metric).

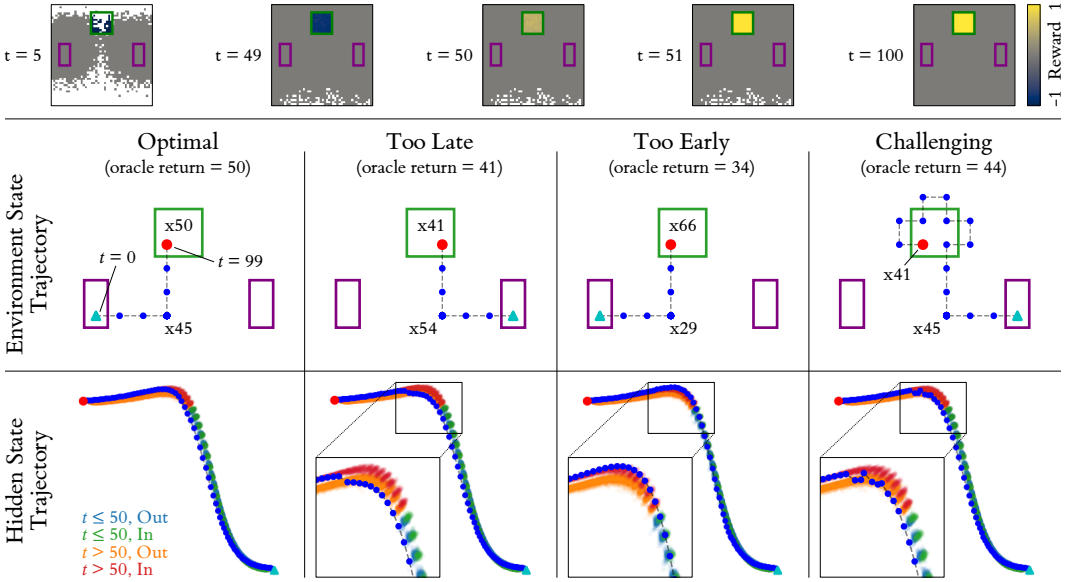

Figure A2: Timer task trajectory probing.
Top: Predicted reward with respect to time and position. We observe that the model has correctly captured the transition from negative to positive reward in the treasure region at $t = 50$, with no reward outside of this region. Although the reward is positive at $t = 50$, the model is uncertain at this point, i.e., the transition from negative to positive reward happens over two timesteps rather than one.
Middle/bottom: Four trajectory probes demonstrating the model's hidden state transitions. "xn" labels indicate the agent remaining in a position for n timesteps.
Optimal: The agent moves into the treasure region at $t = 50$ and remains there, receiving the maximum possible reward.
Too Late: The agent moves into the treasure region a while after the treasure has already become positive, i.e., it is missing out on reward by not being in the region for as long as possible. This is reflected in the hidden state plot, where the state transitions from the orange to the red region after the $t = 50$ boundary.
Too Early: The agent moves into the treasure region before the treasure becomes positive, therefore, while it earns the maximum amount of positive reward, it also earns negative reward, leading to a sub-optimal result.
Challenging: The agent moves into the treasure region at the correct time, but proceeds to jump in and out of the treasure region before settling, leading to lost reward. The hidden state trajectories somewhat mimic this movement by transitioning between the orange and red regions, although the jumps are less clear near to the $t = 50$ transition point, suggesting the model is uncertain at this point.

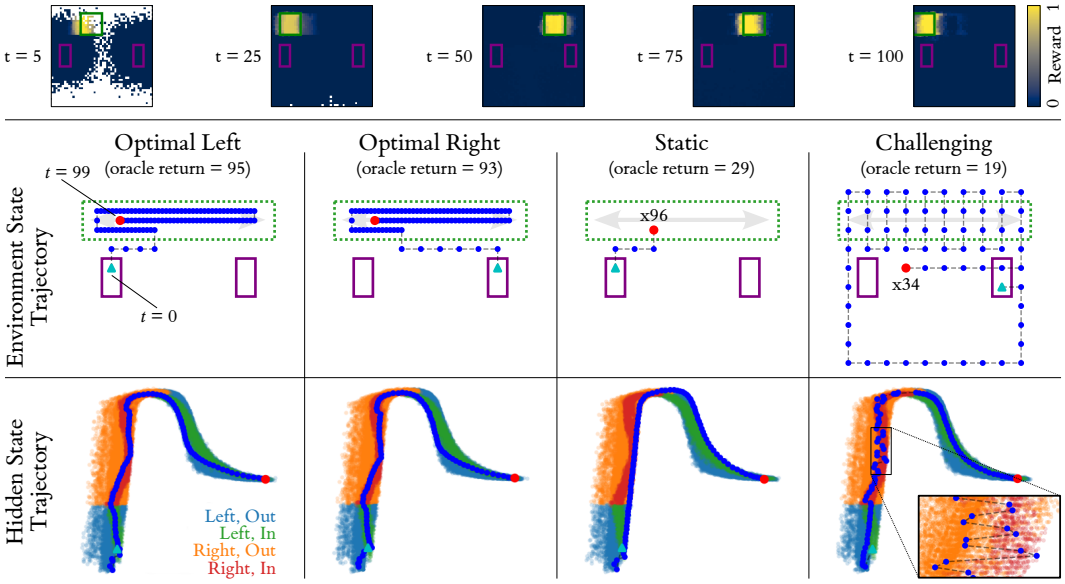

Figure A3: Moving task trajectory probing.
Top: The predicted reward with respect to time (and thus implicitly, the position of the treasure region). The model has learnt to track the treasure region as it moves, although there is noise around the left and right edges of the region, highlighting the difficulty of recovering the treasure region's horizontal position exactly.
Middle/bottom: Four trajectory probes showing the model's hidden state transitions. Note the green dotted region indicates the overall boundary of the treasure region, i.e., the treasure lies somewhere within that boundary, with its true horizontal position dependent on time. "xn" labels indicate the agent remaining in a position for n timesteps.
Optimal Left: The agent moves within the treasure region as quickly as possible and then moves with the treasure (keeping within it) for the remainder of the episode. The hidden state trajectories follow the "In" regions (green and red).
Optimal Right: A similar optimal probe where the agent starts from the right-hand spawn zone rather than the left. In this case, the agent has further to move before moving into the treasure region, leading to slightly less overall reward than when it starts from the left.
Static: Rather than moving with the treasure region, the agent stays still and allows the treasure to pass over it, gaining reward for some timesteps but not others. We observe that the hidden state trajectory also reflects this — the state transitions between the "In" and "Out" regions.
Challenging: The agent takes a while to move towards the treasure region, and then passes in and out of the boundary in which the treasure resides, picking up some reward. We can see from the hidden state trajectory that this motion is captured in the hidden states — the trajectory transitions between the orange and red regions as the agent passes back and forth through the treasure region.

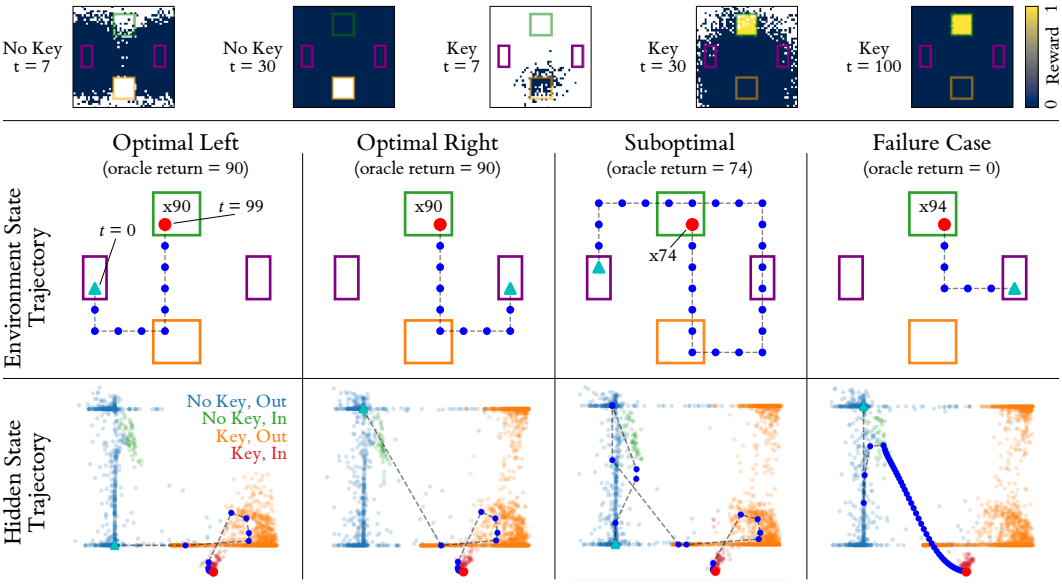

Figure A4: Key task trajectory probing.
Top: The predicted reward with respect to time and collection of the key. The model has learnt to only give reward when the agent is in the treasure region after collecting the key.
Middle/bottom: Four trajectory probes showing the model's hidden state transitions. "xn" labels indicate the agent remaining in a position for n timesteps.
Optimal Left: The agent collects the key as quickly as possible, and then proceeds to move into the treasure region and wait there until the end of the episode. The hidden state trajectory shows two notable transitions, first when the key is collected (blue to orange) and then when the agent enters the treasure region (orange to red).
Optimal Right: A similar optimal policy, but from the right-hand spawn zone rather than the left. Note the hidden state trajectory is very similar, apart from the initial hidden state, which is at the opposite side of the blue region, representing the different spawn position.
Suboptimal: The agent passes through the treasure region before collecting the key, and then follows an optimal policy. We observe a transition in the hidden state trajectory from the blue to the green region that corresponds to the agent's premature entrance into the treasure region.
Failure case: The agent enters the treasure region without collecting the key and remains there for the rest of the episode. This highlights a failure case of the model, where the hidden state "drifts" from the green region to the red region, i.e., the model convinces itself that the agent must have picked up the key at some point. Such *causal confusion* errors are well-documented in the RM literature [40]. In our case, we suspect that the error is due to a bias in our data generation method inducing a correlation between key possession and time spent in the treasure region. As described in Appendix E.2, we specifically screen for trajectories where the treasure is visited with the key (the "treasure" class) but not for those where it is visited without it. There are thus likely to be very few training trajectories that spend a lot of time in the treasure region without collecting the key, making this probe an extreme outlier on which performance is poor. Adding and selecting for a fourth "key_no_treasure" class during data generation may have mitigated this issue.

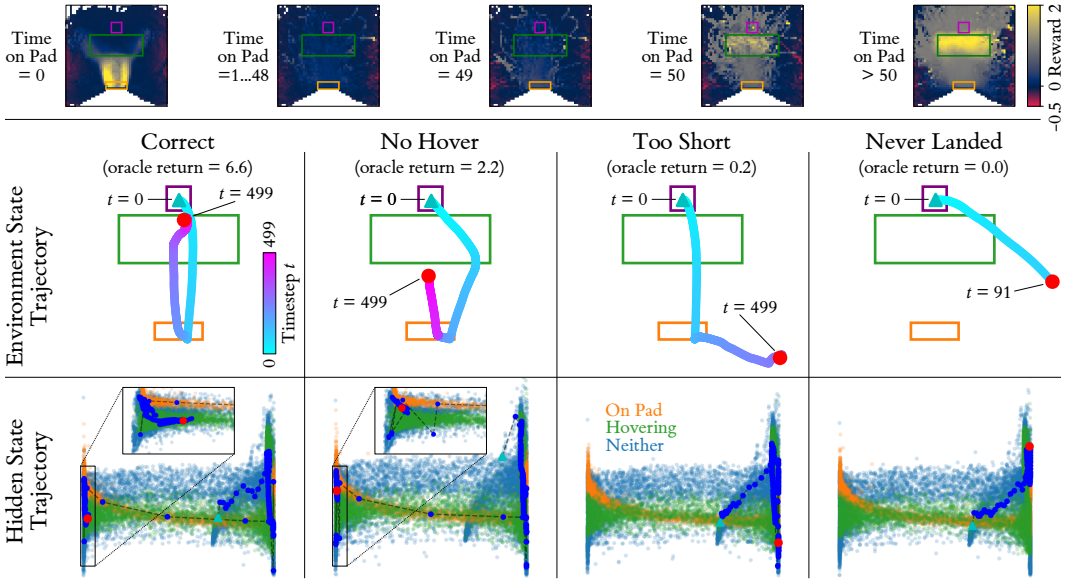

Figure A5: Lunar Lander task trajectory probing.
Top: The predicted reward with respect to the agent's position and the length of time that it has been on the pad. The model has learnt to reward the agent for landing on the pad when it has not previously landed, and then reward it for taking off and remaining in the hover zone after 50 timesteps on the pad, as per the oracle. However, observe how the top half of the hover zone is attributed higher reward than the lower half. This feature is not present in the oracle (which rewards the entire hover zone equally) but makes great practical sense, as a stochastic policy is less likely to drop out of the zone under the effect of gravity if it remains some distance from the bottom edge. Also note the small negative rewards near the environment boundaries, which disincentivise positions with a high risk of leading to early termination, and the "funnel" of intermediate positive reward, which may help to guide the agent up from the pad to the hover zone. Collectively, these deviations from the oracle reward function could actually be seen as *improvements*, and explain why we observe significantly higher performance on the R_{hover} reward component when the CSC Instance Space LSTM reward model is used, compared with using the oracle itself.
Middle/bottom: Four trajectory probes showing the model's hidden state transitions.
Correct: The agent lands on the pad as quickly as possible, waits for 50 timesteps, and then takes off again and hovers in the hover zone. The hidden state trajectory shows a transition from right to left once the agent has been on the pad for long enough. It also clearly shows the agent is in the Hovering hidden state region.
No Hover: In this trajectory, the agent remains on the pad for too long and does not enter the hover zone. It has a similar hidden state transition to the *Correct* trajectory but does not enter the Hovering hidden state region.
Too Short: The agent lands on the pad but does not stay there for the required 50 timesteps. In this particular case, the agent lands correctly but then slips out of the landing pad, coming to rest on a different area of terrain until the end of the trajectory. In the hidden state trajectory, there is no transition from the right side to the left side, matching the idea that the agent has not remained on the pad for long enough.
Never Landed: The agent does not land on the pad at all. In this particular example, the agent reaches the boundary of the environment, which causes the episode to terminate early. Similar to the *Too Short* example, the hidden state trajectory does not transition from right to left.