

Robust Reward Machine Learning and Execution in Noisy Environments

ROKO PARAC, Imperial College London, United Kingdom

LORENZO NODARI, University of Brescia, Italy

DANIEL FURELOS-BLANCO, Imperial College London, United Kingdom

LEO ARDON, Imperial College London, United Kingdom

FEDERICO CERUTTI, University of Brescia, Italy

ALESSANDRA RUSSO, Imperial College London, United Kingdom

Background: Reward machines (RMs) represent non-Markovian reward functions as finite-state machines over high-level propositional events. Traditionally, these events are generated by a deterministic labelling function that maps environment states to proposition sets. However, real-world labelling functions are inherently noisy.

Objectives: Design and implement a method that learns RMs in the presence of noisy labelling functions and exploits them for policy learning in both discrete and continuous environments.

Methods: We propose PROB-IRM, a method that learns RMs from noisy proposition histories using inductive logic programming (ILP) and exploits them for policy learning. Noisy proposition histories are represented as training examples for ILP, which learns an RM that maximises coverage of low-uncertainty histories. RM learning is interleaved with policy optimisation. PROB-IRM uses a belief over RM states to determine when to relearn the RM and to provide additional guiding rewards.

Results: PROB-IRM is evaluated on environments with discrete and continuous state spaces under varying levels of labelling noise. PROB-IRM performs comparably to baselines using handcrafted RMs and outperforms approaches that use recurrent neural networks to encode history, despite operating with noisy labels.

Conclusions: PROB-IRM represents a step towards deploying reward machines in noisy, real-world environments.

JAIR Associate Editor: Marek Petrik

JAIR Reference Format:

Roko Parac, Lorenzo Nodari, Daniel Furelos-Blanco, Leo Ardon, Federico Cerutti, and Alessandra Russo. 2026. Robust Reward Machine Learning and Execution in Noisy Environments. *Journal of Artificial Intelligence Research* 87, Article 18 (October 2026), 44 pages. DOI: [10.1613/jair.1.22539](https://doi.org/10.1613/jair.1.22539)

1 Introduction

Reinforcement learning (RL; [Sutton and Barto 2018](#)) is a machine learning paradigm that models how agents learn to make decisions by interacting with an environment, improving their behaviour through experience. The objective of RL is to learn a policy that maximises cumulative reward over time. RL has played a central role in several landmark AI advances, including AlphaGo ([Silver et al. 2016](#)) and the training of large language models ([Ouyang et al. 2022](#)). However, despite these successes, challenges remain in extending RL to a broader

Authors' Contact Information: Roko Parac, ORCID: [0009-0009-2916-2410](https://orcid.org/0009-0009-2916-2410), roko.parac18@ic.ac.uk, Imperial College London, London, United Kingdom; Lorenzo Nodari, ORCID: [0009-0008-7997-8170](https://orcid.org/0009-0008-7997-8170), lorenzo.nodari@unibs.it, University of Brescia, Brescia, Italy; Daniel Furelos-Blanco, ORCID: [0000-0001-7461-1910](https://orcid.org/0000-0001-7461-1910), daniel.furelos.blanco@gmail.com, Imperial College London, London, United Kingdom; Leo Ardon, ORCID: [0000-0003-4400-7127](https://orcid.org/0000-0003-4400-7127), leo.ardon19@ic.ac.uk, Imperial College London, London, United Kingdom; Federico Cerutti, ORCID: [0000-0003-0755-0358](https://orcid.org/0000-0003-0755-0358), federico.cerutti@unibs.it, University of Brescia, Brescia, Italy; Alessandra Russo, ORCID: [0000-0002-3318-8711](https://orcid.org/0000-0002-3318-8711), a.russo@ic.ac.uk, Imperial College London, London, United Kingdom.



This work is licensed under a [Creative Commons Attribution International 4.0 License](https://creativecommons.org/licenses/by/4.0/).

© 2026 Copyright held by the owner/author(s).

DOI: [10.1613/jair.1.22539](https://doi.org/10.1613/jair.1.22539)

range of real-world applications (Dulac-Arnold et al. 2021). Key challenges include operating robustly under partial observability and noisy observations, generalising across tasks, and improving policy interpretability.

Reward machines (RMs; Toro Icarte, Klassen, R. A. Valenzano, et al. 2018) constitute an effective method for addressing some of these challenges. They are finite-state machines that represent non-Markovian reward functions in terms of propositional high-level events abstracting environment states. An RM serves as memory for the agent, decomposing a complex task into a structured sequence of intermediate sub-tasks. By augmenting the environment state space with RM states, the reward function becomes Markovian, enabling the application of standard RL algorithms. The structure of an RM is interpretable and, when combined with appropriate techniques such as reward shaping or hierarchical RL, can mitigate sparse reward challenges and facilitate policy transfer across related tasks.

Recent studies (Furelos-Blanco, Law, Jonsson, et al. 2021; M. Hasanbeig, Jeppu, et al. 2021; Toro Icarte, Waldie, et al. 2019; Xu et al. 2020) have introduced methods for learning RMs, reducing manual engineering effort and enhancing their applicability across tasks. However, most assume an oracle that perfectly detects high-level events; that is, a *perfect labelling function* mapping environment states to propositions. This assumption is a key limitation, as real-world event detection through sensors or AI models is rarely perfect due to technological and environmental constraints. Crucially, any errors in detecting high-level events lead to learning incorrect RMs.

Our goal is to devise a method that handles noisy sensor detections when learning and exploiting RMs for non-Markovian reward tasks. We introduce **PROB-IRM (Probabilistic Induction of Reward Machines)**, the first method that directly learns and exploits RMs when the labelling function is *noisy*. The agent’s belief about event occurrences is modelled using a Bayesian posterior conditioned on sensor outputs. To learn RMs from noisy traces, PROB-IRM uses ILASP (Law et al. 2015), a state-of-the-art inductive logic programming system capable of learning from noisy examples. Pivotal to PROB-IRM is the interleaving of RM and policy learning: a new RM is learned whenever the RL agent generates a trace not believed to be accepted by the current RM. To accelerate policy learning, PROB-IRM employs a probabilistic reward shaping formulation based on posterior beliefs derived from the traces. This paper extends our prior work (Parac et al. 2024) with the following novel contributions:

- *Complete ILASP encoding for learning RMs from noisy traces.* We present the ILASP encoding for the RM learning task, clearly identifying the changes needed compared to learning from noise-free traces (Furelos-Blanco, Law, Jonsson, et al. 2021).
- *Enhanced generality and scalability.* Our prior work used tabular action-value functions, requiring the discretisation of RM state beliefs and limiting the applicability to discrete state spaces. We address this by approximating policies with neural networks trained using Proximal Policy Optimisation (PPO; Schulman et al. 2017), enabling seamless integration with continuous domains and improved robustness (Yu et al. 2022). We also modified the architecture to enable parallel trace collection, significantly improving scalability.
- *Extensive evaluation and ablation study.* We evaluate PROB-IRM on tasks with discrete and continuous state spaces and sparse non-Markovian rewards, providing in-depth ablation analysis of design choices. We compare against baselines using handcrafted RMs and a fully neural baseline (RECURRENT PPO; Raffin et al. 2021; Schulman et al. 2017). Results demonstrate that across diverse noise configurations, PROB-IRM performs comparably to the handcrafted-RM baselines and outperforms RECURRENT PPO.

The paper is organised as follows. Section 2 provides background on RL, RMs, and ILASP. Section 3 formalises the noisy event detection setting used throughout. Section 4 describes our method for learning RMs from noisy traces. Section 5 discusses RM exploitation, including probabilistic reward shaping. Section 6 explains how we interleave RM and policy learning. Section 7 presents experimental results and ablation studies. Section 8 discusses related work, Section 9 highlights limitations, and Section 10 concludes the paper and proposes suggestions for future work.

2 Background

This section provides background on reinforcement learning, reward machines, and inductive logic programming. We first establish notation: given a set X , $\Delta(X)$ is the probability simplex over X , X^* denotes (possibly empty) sequences of elements from X , X^+ denotes non-empty sequences of elements from X , and 2^X is the power set of X . We use $\oplus$ to denote the concatenation of two sequences. The complete notation used throughout the paper is given in Appendix A.

2.1 Reinforcement Learning

We formalise RL tasks as *labelled Markov decision processes* (Fu and Topcu 2014; Furelos-Blanco, Law, Jonsson, et al. 2023).

Definition 2.1 (Labelled Markov decision process). A labelled Markov decision process (MDP) is a tuple $\langle \mathcal{S}, s_0, \mathcal{A}, p, r, \tau, \gamma, \mathcal{P}, \mathcal{L} \rangle$ where:

- $\mathcal{S}$ is a set of states,
- $s_0 \in \mathcal{S}$ is an initial state,
- $\mathcal{A}$ is a set of actions,
- $p : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is a probability transition function,
- $r : (\mathcal{S} \times \mathcal{A})^+ \times \mathcal{S} \rightarrow \mathbb{R}$ is a reward function,
- $\tau : (\mathcal{S} \times \mathcal{A})^* \times \mathcal{S} \rightarrow \{\perp, \top\} \times \{\perp, \top\}$ is a termination function,
- $\gamma \in [0, 1)$ is a discount factor,
- $\mathcal{P}$ is a finite set of *propositions* representing high-level events, and
- $\mathcal{L} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow 2^{\mathcal{P}}$ is a (*perfect*) *labelling function* mapping state-action-state triplets to sets of propositions in $\mathcal{P}$. We refer to these sets as *labels*.

The transition function p is Markovian, whereas the reward function r and termination function τ are not (i.e., they are history-dependent). The termination function τ returns a pair $\langle \tau^T, \tau^G \rangle$, where τ^T indicates whether the current history is terminal, and τ^G indicates whether the goal is achieved.

Given a state-action *history* $h_t = \langle s_0, a_0, \dots, s_t \rangle \in (\mathcal{S} \times \mathcal{A})^* \times \mathcal{S}$, a *trace* $\lambda_t = \langle \mathcal{L}(\emptyset, \emptyset, s_0), \dots, \mathcal{L}(s_{t-1}, a_{t-1}, s_t) \rangle \in (2^{\mathcal{P}})^+$ assigns a label to each transition in h_t , where $\mathcal{L}(\emptyset, \emptyset, s_0)$ represents the initial state label. The goal is to find a *policy* $\pi : (2^{\mathcal{P}})^+ \times \mathcal{S} \rightarrow \Delta(\mathcal{A})$ that maps trace-state pairs to a probability distribution over actions that maximises the expected cumulative discounted reward (or *return*) $R_t = \mathbb{E}_\pi [\sum_{k=t}^n \gamma^{k-t} r(h_{k+1})]$, where n is the episode's last step. The labelling function must preserve sufficient information from histories such that reward and termination can be determined from the corresponding traces. If so, the reward and termination functions can be written in terms of traces instead of history as $r(h_{t+1}) = r(\lambda_{t+1}, s_{t+1})$ and $\tau(h_t) = \tau(\lambda_t, s_t)$, respectively.

The agent-environment interaction is as follows. At time t , the (label) trace is $\lambda_t \in (2^{\mathcal{P}})^+$ and the agent observes a tuple $\langle s_t, \tau_t^T, \tau_t^G \rangle$, where $s_t \in \mathcal{S}$ is the state, and $(\tau_t^T, \tau_t^G) = \tau(\lambda_t, s_t)$ is the termination information with $\tau_t^T \in \{\perp, \top\}$ and $\tau_t^G \in \{\perp, \top\}$ indicating termination and goal status, respectively. The agent also observes a label $L_t = \mathcal{L}(s_{t-1}, a_{t-1}, s_t)$. If the history is non-terminal, the agent takes an action $a_t \in \mathcal{A}$, and the environment transitions to state $s_{t+1} \sim p(\cdot \mid s_t, a_t)$. The agent then observes a new tuple $\langle s_{t+1}, \tau_{t+1}^T, \tau_{t+1}^G \rangle$ and label L_{t+1} , extends the trace as $\lambda_{t+1} = \lambda_t \oplus L_{t+1}$, and receives reward $r(\lambda_{t+1}, s_{t+1})$.

Traces can be classified based on the terminal and goal indicators at time t : a trace λ_t is a *goal trace* if $\langle \tau_t^T, \tau_t^G \rangle = \langle \top, \top \rangle$, a *dead-end trace* if $\langle \tau_t^T, \tau_t^G \rangle = \langle \top, \perp \rangle$, and an *incomplete trace* if $\tau_t^T = \perp$.

EXAMPLE 1. The OFFICEWORLD (Toro Icarte, Klassen, R. A. Valenzano, et al. 2018) domain, illustrated in Figure 1 (left), is a 12×9 grid labelled with designated special locations. At each step, the agent $\mathfrak{A}$ observes its current position in the grid and moves in one of the four cardinal directions; that is, $\mathcal{S} = \{0, \dots, 11\} \times \{0, \dots, 8\}$ and

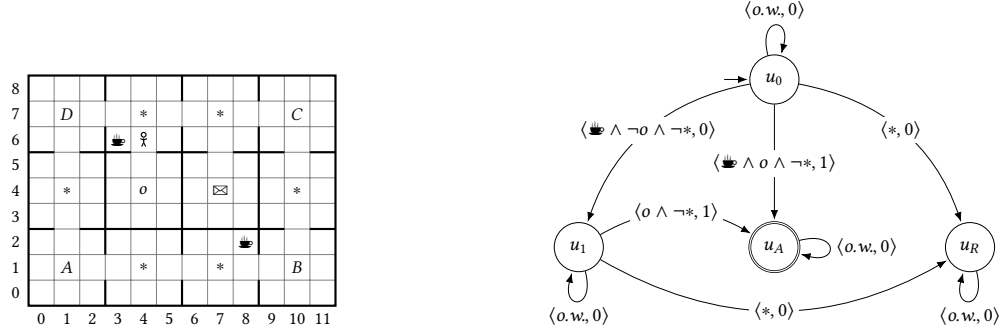


Fig. 1. An OFFICEWORLD instance (left) and a reward machine for the COFFEE task (right), where o.w. stands for *otherwise*.

$\mathcal{A} = \{\text{up, down, left, right}\}$. The agent always moves in the intended direction and remains in place when attempting to move into a wall. The set of propositions $\mathcal{P} = \{\text{☕}, \text{✉}, \text{o}, \text{A}, \text{B}, \text{C}, \text{D}, *\}$ consists of the special locations. The labelling function maps a $\langle s, a, s' \rangle \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ triplet to the set of propositions observed in s' , e.g., $\mathcal{L}(\langle 4, 6 \rangle, \text{left}, \langle 3, 6 \rangle) = \{\text{☕}\}$. Let us consider the COFFEE task, which consists of going to the coffee machine (☕) and then to the office (o). A reward of 1 is obtained for completing the task; otherwise, the reward is 0. Rewards are non-Markovian since the current state (i.e., position on the grid) alone cannot determine the reward. The history $h_4 = \langle \langle 4, 6 \rangle, \text{left}, \langle 3, 6 \rangle, \text{right}, \langle 4, 6 \rangle, \text{down}, \langle 4, 5 \rangle, \text{down}, \langle 4, 4 \rangle \rangle$ yields a reward of 1 and is mapped to the goal trace $\lambda_4 = \langle \{\}, \{\text{☕}\}, \{\}, \{\}, \{\text{o}\} \rangle$.

2.2 Reward Machines

Reward machines provide a structured representation of reward functions by encoding task structure as a finite-state machine over propositional events.

Definition 2.2 (Reward machine). A reward machine (RM; Toro Icarte, Klassen, R. A. Valenzano, et al. 2022, 2018) is a tuple $\mathcal{RM} = \langle U, \mathcal{P}, \delta^u, \delta^r, u_0, u_A, u_R \rangle$, where:

- U is a set of states,
- $\mathcal{P}$ is a set of propositions forming the RM's alphabet,
- $\delta^u : U \times 2^{\mathcal{P}} \rightarrow U$ is a state-transition function,
- $\delta^r : U \times U \rightarrow \mathbb{R}$ is a reward-transition function,
- $u_0, u_A, u_R \in U$ are the initial, accepting, and rejecting states, respectively.

The accepting state indicates successful task completion, while the rejecting state indicates unrecoverable failure.

EXAMPLE 2. Figure 1 (right) shows the RM for the COFFEE task in OFFICEWORLD introduced in Example 1. Edges are labelled with propositional formulas over $\mathcal{P} = \{\text{☕}, \text{✉}, \text{o}, \text{A}, \text{B}, \text{C}, \text{D}, *\}$ and associated transition rewards. To verify whether a formula is satisfied by a label $L \in 2^{\mathcal{P}}$, L is used as a truth assignment: propositions contained in the label are true, and all others are false. For example, the label $\{\text{☕}\}$ satisfies the formula $\text{☕} \wedge \neg \text{o} \wedge \neg *$.

During the agent-environment interaction, the agent tracks its current state in the RM alongside its environment state. Starting from the RM's initial state u_0 , the agent transitions through RM states according to the state-transition function δ^u based on observed labels, receiving rewards via the reward-transition function δ^r .

Given an RM $\mathcal{RM}$ and a trace $\lambda = \langle L_0, \dots, L_n \rangle$, a traversal $\mathcal{RM}(\lambda) = \langle v_0, v_1, \dots, v_{n+1} \rangle$ is the unique sequence of RM states such that $v_0 = u_0$, and $v_{i+1} = \delta^u(v_i, L_i)$ for $i = 0, \dots, n$. Traversals of goal traces terminate in u_A , traversals of dead-end traces terminate in u_R , and traversals of incomplete traces terminate in neither.

EXAMPLE 3. Given the RM in Figure 1 (right) and the goal trace $\lambda = \langle \{\}, \{\bullet\}, \{\}, \{\}, \{o\} \rangle$, the traversal is $\langle u_0, u_0, u_1, u_1, u_1, u_A \rangle$. As expected for a goal trace, the traversal terminates in the accepting state.

To verify that an RM correctly encodes task structure, we check whether its traversals align with trace categories. An RM $\mathcal{RM}$ is *valid* with respect to trace $\lambda = \langle L_0, \dots, L_n \rangle$ if

- $v_{n+1} = u_A$ and the trace λ is a *goal* trace,
- $v_{n+1} = u_R$ and the trace λ is a *dead-end* trace, or
- $v_{n+1} \notin \{u_A, u_R\}$ and the trace λ is an *incomplete* trace,

where v_{n+1} is the last RM state in the traversal $\mathcal{RM}(\lambda)$.

Reward machines are compact trace representations: each RM state encodes how far the agent has advanced toward the goal. Consequently, the reward function becomes Markovian over the joint space of MDP and RM states $\mathcal{S} \times \mathcal{U}$. This enables the application of standard RL algorithms such as Q-learning (Watkins and Dayan 1992), DDPG (Lillicrap et al. 2016), and PPO (Schulman et al. 2017). These algorithms have been successfully applied to RM-based learning in prior work (e.g., Ardon, Furelos-Blanco, Parać, et al. 2025; Li, Z. Chen, Klassen, et al. 2024; Toro Icarte, Klassen, R. A. Valenzano, et al. 2022, 2018).

2.3 Learning ASP Programs from (Noisy) Examples

Our work follows the methodology proposed by Furelos-Blanco, Law, Jonsson, et al. (2021) for learning RMs using *inductive logic programming* (ILP; Muggleton 1991). Specifically, we encode RMs using *answer set programming* (ASP; Lifschitz 2019) and learn them using ILASP (Law et al. 2015), a state-of-the-art ILP system. We briefly introduce ASP and ILASP’s learning framework below.

2.3.1 Answer Set Programming (ASP). ASP is a declarative programming paradigm for knowledge representation and reasoning. In ASP, a problem is encoded as a logic program, and the solutions correspond to the models of that program, called answer sets.

Our ASP programs consist of normal rules, constraints, and facts. *Normal rules* have the form $h : -b_1, \dots, b_n$, where the *head* h is an atom and the *body* $b_1, \dots, b_n$ consists of literals. A literal is either an atom a or its negation $\neg a$. *Constraints* are rules of the form $: -b_1, \dots, b_n$. *Facts* are rules with an empty body, written as h .

Given a set of ground atoms (or interpretation) I , a ground normal rule is satisfied if the head is satisfied by I whenever all body literals are satisfied by I . A ground constraint is satisfied if its body is not satisfied by I . Facts are always satisfied.

An *answer set* is defined via the reduct. The *reduct* P^I of a ground program P with respect to an interpretation I is constructed by removing from P all rules containing $\neg a$ where $a \in I$, and removing all $\neg a$ literals where $a \notin I$. Then, I is an *answer set* of P if and only if I satisfies the rules in P^I and no proper subset of I does.

EXAMPLE 4. Consider the following ASP program:

$$P = \left\{ \begin{array}{l} a : -b, \text{ not } c. \\ b. \end{array} \right\}.$$

For the interpretation $I = \{a, b\}$, the reduct P^I is:

$$\left\{ \begin{array}{l} a : -b. \\ b. \end{array} \right\}.$$

Since I satisfies all rules in P^I and no proper subset of I (e.g., $\{a\}$, $\{b\}$, and $\emptyset$) does, I is the unique answer set of P .

2.3.2 Learning from Noisy Answer Sets. *Learning from answer sets* (LAS; Law et al. 2020) is a paradigm for learning ASP programs from examples. We focus on learning from *noisy examples*, where each example is weighted with a penalty. The goal is to find an ASP program (called a *hypothesis*) that minimises a score that accounts for program complexity (e.g., number of literals) and the penalties incurred by uncovered examples.

Examples are *weighted context-dependent partial interpretations* (WCDPIs; Law et al. 2018), each a tuple $\langle e_{id}, e_{pen}, \langle e_{inc}, e_{exc} \rangle, e_{ctx} \rangle$, where e_{id} is a unique identifier; e_{pen} is a penalty; $\langle e_{inc}, e_{exc} \rangle$ is a pair of atom sets known as a *partial interpretation*; and e_{ctx} is an ASP program known as the *context*, providing example-specific information. An ASP program P covers a WCDPI $\langle e_{id}, e_{pen}, \langle e_{inc}, e_{exc} \rangle, e_{ctx} \rangle$ if and only if there is an answer set A of $P \cup e_{ctx}$ such that $e_{inc} \subseteq A$ and $e_{exc} \cap A = \emptyset$.

These WCDPI examples are used within a LAS task that also specifies background knowledge and the space of learnable rules. We formalise LAS tasks (Law et al. 2018) as follows.

Definition 2.3 (LAS task). A LAS task T is a tuple of the form $\langle B, S_M, E \rangle$, where B is an ASP program known as background knowledge, S_M is a hypothesis space defining the set of learnable ASP rules, and E is a set of WCDPI examples. Given a hypothesis $H \subseteq S_M$,

- (1) $uncov(H, T)$ is the set of all examples $e \in E$ such that H does not cover e ;
- (2) the penalty of H , denoted as $pen(H, T)$, is the sum $\sum_{e \in uncov(H, T)} e_{pen}$;
- (3) the length of H , denoted as $length(H)$, is the sum $\sum_{r \in H} length(r)$, where $length(r) \in \mathbb{N}_{>0}$;
- (4) the score of H , denoted as $\mathcal{S}(H, T)$, is the sum $length(H) + pen(H, T)$;
- (5) H is an inductive solution of T iff $\mathcal{S}(H, T)$ is finite; and
- (6) H is an *optimal inductive solution* of T iff $\mathcal{S}(H, T)$ is finite and $\nexists H' \subseteq S_M$ such that $\mathcal{S}(H, T) > \mathcal{S}(H', T)$.

Examples with infinite penalty ($e_{pen} = \infty$) are non-noisy and must be covered; finite penalties allow trading off coverage for simpler hypotheses. We use ILASP (Law et al. 2015), a state-of-the-art system for learning ASP programs from noisy examples.

EXAMPLE 5. Consider the LAS tasks $T_1 = \langle B, S_M, E_1 \rangle$ and $T_2 = \langle B, S_M, E_2 \rangle$, where:

$$B = \{b : -c.\}, \quad S_M = \left\{ \begin{array}{l} a. \\ :- b. \\ :- \text{not } b. \\ a : - b. \\ a : - \text{not } b. \end{array} \right\}, \quad E_1 = \left\{ \begin{array}{l} \langle ex1, \infty, \langle \{a\}, \emptyset \rangle, \{c.\} \rangle, \\ \langle ex2, \infty, \langle \emptyset, \{a\} \rangle, \emptyset \rangle \end{array} \right\},$$

$$E_2 = \left\{ \begin{array}{l} \langle ex1, 1, \langle \{a\}, \emptyset \rangle, \{c.\} \rangle, \\ \langle ex2, 1, \langle \emptyset, \{a\} \rangle, \emptyset \rangle \end{array} \right\}.$$

The optimal inductive solution for T_1 is $H_1 = \{a : -b.\}$, with score $\mathcal{S}(H_1, T_1) = 2$ since $pen(H_1, T_1) = 0$ (all examples are covered) and $length(H_1) = 2$.

For T_2 , the examples are identical but have finite penalties. The empty hypothesis $H_2 = \emptyset$ achieves a better score: $\mathcal{S}(H_2, T_2) = 1$ since $pen(H_2, T_2) = 1$ ($ex1$ is uncovered) and $length(H_2) = 0$. While this example yields a trivial solution, it illustrates the key principle: when examples have finite penalties, the learner can sacrifice coverage of low-confidence examples to find simpler and more general hypotheses.

3 Noisy Labelled Markov Decision Processes

The labelled MDP defined in Section 2 assumes a perfect labelling function $\mathcal{L}$. In practice, however, event detection through sensors or learned models is inherently noisy. We formalise this setting through a *noisy labelled MDP*, where the key change is replacing the perfect labelling function with a probabilistic one.

Definition 3.1 (Noisy labelled Markov decision process). A noisy labelled Markov decision process is a tuple $\langle \mathcal{S}, s_0, \mathcal{A}, p, r, \tau, \gamma, \mathcal{P}, \tilde{\mathcal{L}} \rangle$ where $\mathcal{S}, s_0, \mathcal{A}, p, r, \tau, \gamma, \mathcal{P}$ are as in Definition 2.1, and $\tilde{\mathcal{L}}$ is a noisy labelling function.

The noisy labelling function maps each transition to a probability distribution over propositions, capturing sensor uncertainty. We use $l = \top$ and $l = \perp$ to denote the actual presence and absence of proposition l , and $\tilde{l} = \top$ and $\tilde{l} = \perp$ to denote sensor detection and non-detection, respectively.

Definition 3.2 (Noisy labelling function). The noisy labelling function $\tilde{\mathcal{L}}$ maps each transition to the probability of each proposition:

$$\tilde{\mathcal{L}}(s_t, a_t, s_{t+1}) = \{\langle l, [\tilde{\mathcal{L}}(s_t, a_t, s_{t+1})]_l \rangle \mid l \in \mathcal{P}\},$$

where $[\tilde{\mathcal{L}}(s_t, a_t, s_{t+1})]_l = P(l = \top \mid s_t, a_t, s_{t+1})$ is the probability of observing l on transition $\langle s_t, a_t, s_{t+1} \rangle$.

For compactness, we write pairs as $l : p, \dots$ rather than $\langle l, p \rangle$, omitting propositions with probability zero. For instance, if $[\tilde{\mathcal{L}}(s_0, a_0, s_1)]_{\text{p}} = 0.65$ and $[\tilde{\mathcal{L}}(s_0, a_0, s_1)]_o = 0.01$, with all other propositions having probability zero, we write $\tilde{\mathcal{L}}(s_0, a_0, s_1) = \{\text{p} : 0.65, o : 0.01\}$.

A *noisy trace* is the probabilistic analogue of a trace, where each label is replaced by its probability distribution.

Definition 3.3 (Noisy trace). Given a state-action history $h_t = \langle s_0, a_0, s_1, a_1, \dots, s_t \rangle$, the noisy trace $\tilde{\lambda}_t$ is:

$$\tilde{\lambda}_t = \langle \tilde{\mathcal{L}}(\emptyset, \emptyset, s_0), \tilde{\mathcal{L}}(s_0, a_0, s_1), \dots, \tilde{\mathcal{L}}(s_{t-1}, a_{t-1}, s_t) \rangle.$$

We make the following assumptions about this setting:

ASSUMPTION 1 (CONDITIONAL INDEPENDENCE). Given a transition $\langle s_t, a_t, s_{t+1} \rangle$, let $\tilde{\mathcal{P}}_{t+1} \in 2^{\mathcal{P}}$ denote the set of propositions detected by the sensors at this transition. We assume that propositions are conditionally independent given their respective sensor detections. Thus, the probability of each proposition depends only on its own sensor output:

$$P(l = \top \mid s_t, a_t, s_{t+1}) = P(l = \top \mid \tilde{\mathcal{P}}_{t+1}) = P(l = \top \mid \tilde{l}_{t+1}).$$

ASSUMPTION 2 (DETERMINISTIC TERMINATION). The termination function τ remains deterministic, so trace termination is determined with certainty.

ASSUMPTION 3 (REWARD STRUCTURE). The reward function r assigns a reward of 1 to goal traces, -1 to dead-end traces, and 0 to incomplete traces.

Assumption 1 is common in noisy labelling models (Ding et al. 2011; Li, Z. Chen, Klassen, et al. 2024; Verginis et al. 2024). Assumption 2 ensures that, despite noisy labels, the agent knows with certainty when an episode terminates. This is reasonable when termination is determined by a supervising agent (e.g., a human) who can accurately assess task completion. Assumption 3, adopted from prior work on RM learning (e.g., Ardon, Furelos-Blanco, and Russo 2024; Furelos-Blanco, Law, Jonsson, et al. 2023), allows us to focus on learning RM structure rather than rewards.

We now describe one concrete way to obtain the noisy labelling function using a binary sensor model. The agent is equipped with a binary sensor for each proposition $l \in \mathcal{P}$, characterised by its sensitivity and specificity of detection. The *sensitivity* of the sensor is the probability of detection given actual presence: $P(\tilde{l} = \top \mid l = \top)$. The *specificity* is the probability of non-detection given actual absence: $P(\tilde{l} = \perp \mid l = \perp)$. Since the sensor output may not always match the true environment state, the agent computes a posterior belief over proposition occurrence using Bayes' rule:

$$P(l = \top \mid \tilde{l} = y) = \frac{P(\tilde{l} = y \mid l = \top)P(l = \top)}{\sum_{x \in \{\top, \perp\}} P(\tilde{l} = y \mid l = x)P(l = x)}, \quad (1)$$

where $y \in \{\top, \perp\}$ is the sensor outcome, $P(l = \top)$ is the prior probability of proposition l being present, and the probabilities $P(\tilde{l} = \perp \mid l = \top)$, $P(\tilde{l} = \top \mid l = \perp)$ and $P(l = \perp)$ follow from sensitivity, specificity and the prior. We experimentally determine priors using the relative occurrence of the propositions in the environment.

REMARK. The choice of sensor model is orthogonal to the rest of the method. Any model that provides $P(l = \top \mid s_t, a_t, s_{t+1})$ for every $l \in \mathcal{P}$ is a suitable candidate. The binary sensor model with fixed sensitivity and specificity presented here is closely related to that of [Ghasemi et al. \(2020\)](#), with the key distinction that we do not update the sensor model as new information becomes available.

EXAMPLE 6. Consider the COFFEE task from Example 1 with $\mathcal{P} = \{\text{☕}, o\}$ and the history $h = \langle \langle 4, 6 \rangle, \text{left}, \langle 3, 6 \rangle \rangle$. The priors are $P(\text{☕} = \top) = \frac{2}{9 \cdot 12}$ and $P(o = \top) = \frac{1}{9 \cdot 12}$ (two ☕ and one o in a 9×12 grid). Both sensors have sensitivity and specificity 0.99. We compute the posteriors for each step using Equation (1):

- Step $t = 0$. Suppose no propositions are detected ($\tilde{\mathcal{P}}_0 = \emptyset$):

$$\begin{aligned} P(\text{☕} = \top \mid \tilde{\text{☕}} = \perp) &= \frac{P(\tilde{\text{☕}} = \perp \mid \text{☕} = \top)P(\text{☕} = \top)}{\sum_{x \in \{\top, \perp\}} P(\tilde{\text{☕}} = \perp \mid \text{☕} = x)P(\text{☕} = x)} \\ &= \frac{(1 - 0.99) \cdot \frac{2}{9 \cdot 12}}{(1 - 0.99) \cdot \frac{2}{9 \cdot 12} + 0.99 \cdot \left(1 - \frac{2}{9 \cdot 12}\right)} \simeq 0.0002, \\ P(o = \top \mid \tilde{o} = \perp) &= \frac{(1 - 0.99) \cdot \frac{1}{9 \cdot 12}}{(1 - 0.99) \cdot \frac{1}{9 \cdot 12} + 0.99 \cdot \left(1 - \frac{1}{9 \cdot 12}\right)} \simeq 0.0001. \end{aligned}$$

Thus $\tilde{\mathcal{L}}(\emptyset, \emptyset, \langle 4, 6 \rangle) = \{\text{☕} : 0.0002, o : 0.0001\}$.

- Step $t = 1$. Suppose the coffee sensor fires ($\tilde{\mathcal{P}}_1 = \{\text{☕}\}$):

$$\begin{aligned} P(\text{☕} = \top \mid \tilde{\text{☕}} = \top) &= \frac{0.99 \cdot \frac{2}{9 \cdot 12}}{0.99 \cdot \frac{2}{9 \cdot 12} + (1 - 0.99) \cdot \left(1 - \frac{2}{9 \cdot 12}\right)} \simeq 0.6513, \\ P(o = \top \mid \tilde{o} = \perp) &\simeq 0.0001. \end{aligned}$$

Thus $\tilde{\mathcal{L}}(\langle 4, 6 \rangle, \text{left}, \langle 3, 6 \rangle) = \{\text{☕} : 0.6513, o : 0.0001\}$.

The resulting noisy trace is therefore $\hat{\lambda}_1 = \langle \{\text{☕} : 0.0002, o : 0.0001\}, \{\text{☕} : 0.6513, o : 0.0001\} \rangle$.

4 Learning Reward Machines from Noisy Traces

In this section, we present our approach for learning RMs from noisy traces using ILASP. We assume a fixed dataset of noisy traces is given and later describe the interleaved collection and learning process in Section 6.

Figure 2 provides an overview of the pipeline: noisy traces are sampled to produce grounded traces, which are supplemented with traces generated from an existing RM (when available). These traces are converted into ILASP examples with associated penalties, which, together with background knowledge and a hypothesis space, are used by ILASP to learn an RM.

We organise this section as follows. Section 4.1 describes how we generate training examples from noisy traces and existing RMs. Section 4.2 presents the ASP encoding of RMs and traces. Finally, Section 4.3 defines the complete ILASP learning task, including background knowledge, hypothesis space, example construction, and complexity control mechanisms.

4.1 Generating Example Traces

We generate training examples from two sources: grounded traces sampled from noisy traces, and traces derived from an existing RM structure. These are described in the following subsections.

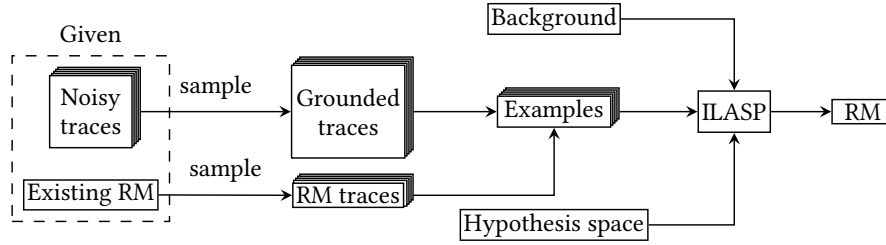


Fig. 2. Overview of the RM learning pipeline. Traces are sampled from noisy traces and from an existing RM, converted into ILASP examples, and used together with background knowledge and a hypothesis space to learn a new RM.

4.1.1 From Noisy Traces. A key challenge in learning RMs from noisy traces lies in mapping these traces to WCDPIs. We use a sampling-based method: at each step in the trace, for each proposition we sample from a Bernoulli distribution parameterised by its probability. This generates multiple grounded traces, each representing a plausible realisation of the noisy trace.

Formally, let $\tilde{\lambda} = \langle \tilde{\mathcal{L}}(\emptyset, \emptyset, s_0), \tilde{\mathcal{L}}(s_0, a_0, s_1), \dots, \tilde{\mathcal{L}}(s_{n-1}, a_{n-1}, s_n) \rangle$ be a noisy trace, and let $\tilde{\lambda}_{i,l} = [\tilde{\mathcal{L}}(s_{i-1}, a_{i-1}, s_i)]_l$ denote the probability of observing proposition l at step i . We generate a *grounded trace* $\lambda' = \langle L_1, \dots, L_n \rangle$ from $\tilde{\lambda}$ by treating each proposition occurrence as a Bernoulli trial: $\lambda'_{i,l} \sim \text{Ber}(\tilde{\lambda}_{i,l})$ for all $l \in \mathcal{P}$, where $L_i = \{l \in \mathcal{P} \mid \lambda'_{i,l} = 1\}$. The process is performed $n_{\lambda'}$ times for each noisy trace, resulting in $n_{\lambda'}$ grounded traces. Following prior work on learning RMs (Furelos-Blanco, Law, Jonsson, et al. 2021; Toro Icarte, Waldie, et al. 2019), we compress the traces by removing the empty labels and, thereafter, removing contiguous identical labels.

EXAMPLE 7. Consider the proposition set $\mathcal{P} = \{\clubsuit, o\}$ and the noisy goal trace $\tilde{\lambda} = \{\{\clubsuit : 0.01, o : 0.01\}, \{\clubsuit : 0.9, o : 0.01\}, \{\clubsuit : 0.9, o : 0.01\}, \{\clubsuit : 0.01, o : 0.01\}, \{\clubsuit : 0.01, o : 0.9\}\}$. Sampling could yield the most likely trace: $\lambda' = \langle \{\}, \{\clubsuit\}, \{\clubsuit\}, \{\}, \{o\} \rangle$. After compression (removing empty labels and consecutive duplicates), $\lambda' = \langle \{\clubsuit\}, \{o\} \rangle$.

Additionally, we generate incomplete examples from prefixes of the compressed goal and dead-end traces, similar to Ardon, Furelos-Blanco, and Russo (2024). This provides the learner with information about valid intermediate traces and helps prevent overgeneralisation.

4.1.2 From an Existing Reward Machine. The examples derived from noisy traces may be insufficient or biased for learning a good RM. This is particularly problematic when traces are collected by a policy that exploits a given RM: such traces concentrate on near-optimal paths and lack diversity. For instance, the agent may only collect goal traces along one path, missing incomplete and dead-end traces that are essential for learning intermediate states and alternative paths. This issue is present when interleaving RM learning with policy optimisation, described in Section 6.

To address this, we generate additional examples from the exploited RM's structure. These examples represent paths the RM considers valid, supplementing the collected traces and ensuring that the RM's knowledge is retained even when corresponding traces become rare. Given an RM, we generate traces as follows:

- (1) Enumerate all paths (i.e., sequences of edge formulas) from u_0 to u_A and u_R .
- (2) For each path, generate all subsequences of formulas.
- (3) Convert each subsequence to a trace by taking the minimal label satisfying each formula. The trace is a goal trace if the traversal ends with u_A , a dead-end trace if it ends with u_R , and an incomplete trace otherwise.

Prior work on learning RMs without noise (e.g., Furelos-Blanco, Law, Jonsson, et al. 2021; M. Hasanbeig, Jeppu, et al. 2021; Toro Icarte, Waldie, et al. 2019; Xu et al. 2020) does not require this technique, as those methods satisfy

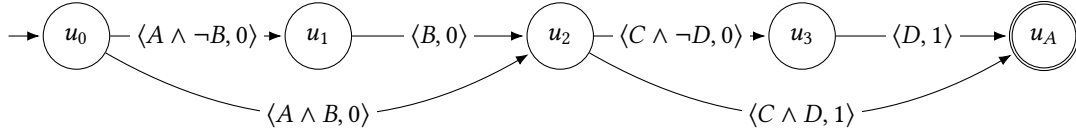


Fig. 3. Reward machine for the VISITABCD task used in Example 8. The agent must go to locations A, B, C and D in order. This RM accounts for co-location: A may appear with B, and C may appear with D.

all examples exactly; hence, infrequent but valid traces are never ignored. In our setting, however, infrequent examples may correspond to noise and receive low penalties, risking the loss of valid but rare behaviours.

EXAMPLE 8. Consider the RM in Figure 3 for the VISITABCD task in OFFICEWORLD:

(1) There are four paths from u_0 to u_A :

$$\langle A \wedge \neg B, B, C \wedge \neg D, D \rangle, \quad \langle A \wedge B, C \wedge \neg D, D \rangle, \quad \langle A \wedge B, C \wedge D \rangle, \quad \langle A \wedge \neg B, B, C \wedge D \rangle.$$

(2) For the path $\langle A \wedge B, C \wedge \neg D, D \rangle$, the subsequences are:

$$\langle A \wedge B, C \wedge \neg D, D \rangle, \quad \langle A \wedge B, C \wedge \neg D \rangle, \quad \langle A \wedge B, D \rangle, \quad \langle C \wedge \neg D, D \rangle, \quad \langle A \wedge B \rangle, \quad \langle C \wedge \neg D \rangle, \quad \langle D \rangle.$$

(3) Each subsequence becomes a trace using minimal labels. For example, $\langle A \wedge B, C \wedge \neg D \rangle$ yields the incomplete trace $\lambda = \{\{A, B\}, \{C\}\}$.

4.2 ASP Representation of Reward Machines and Traces

To learn RMs using ILASP, we encode them as ASP programs. The RMs we learn have the same structure as those defined in Section 2, and we learn them from grounded (deterministic) traces rather than noisy traces. Consequently, we build upon the encoding of Furelos-Blanco, Law, Jonsson, et al. (2021), which was designed for learning from perfect traces. We first introduce notation for expressing RM transitions as propositional logic formulas, then present the ASP encoding.

The state-transition function of an RM can be expressed using a logical transition function $\varphi : U \times U \rightarrow \text{DNF}_{\mathcal{P}}$ (Furelos-Blanco, Law, Jonsson, et al. 2021) that maps state pairs to disjunctive normal form (DNF) formulas over $\mathcal{P}$, where each disjunct corresponds to an edge. The number of disjuncts in the formula between a pair of states $u, u' \in U$ is denoted $|\varphi(u, u')|$.

EXAMPLE 9. For the RM in Figure 1, we have $\varphi(u_0, u_1) = \text{true} \wedge \neg o \wedge \neg *$ with $|\varphi(u_0, u_1)| = 1$.

Definition 4.1 (ASP representation of reward machines). Given an RM $\mathcal{RM} = \langle U, \mathcal{P}, \delta^u, \delta^r, u_0, u_A, u_R \rangle$, we encode it as $M(\mathcal{RM}) = M_U(\mathcal{RM}) \cup M_\varphi(\mathcal{RM})$, where:

$$M_U(\mathcal{RM}) = \{\text{used_state}(u). \mid u \in U\}$$

encodes the RM states with $\text{used_state}(u)$ indicating u is an RM state, and

$$M_\varphi(\mathcal{RM}) = \left\{ \begin{array}{l} \text{ed}(u, u', i). \\ \hat{\varphi}(u, u', i, T) : \neg \text{not prop}(l_1, T), \text{step}(T). \\ \vdots \\ \hat{\varphi}(u, u', i, T) : \neg \text{not prop}(l_n, T), \text{step}(T). \\ \hat{\varphi}(u, u', i, T) : \neg \text{prop}(l_{n+1}, T), \text{step}(T). \\ \vdots \\ \hat{\varphi}(u, u', i, T) : \neg \text{prop}(l_m, T), \text{step}(T). \end{array} \mid \begin{array}{l} u \in U \setminus \{u_A, u_R\}, \\ u' \in U \setminus \{u\}, \\ 1 \leq i \leq |\varphi(u, u')|, \\ \text{disj}_i \in \varphi(u, u'), \\ \text{disj}_i = l_1 \wedge \dots \wedge l_n \\ \wedge \neg l_{n+1} \wedge \dots \wedge \neg l_m \end{array} \right\}$$

encodes transitions with disj_i denoting the i -th disjunct of $\varphi(u, u')$, $\text{ed}(u, u', i)$ representing the i -th edge from u to u' , $\text{prop}(l, T)$ indicating proposition l holds at time T , and $\text{step}(T)$ denoting that T is a timestep.

The encoding uses $\hat{\varphi}$ (negation of φ) rather than φ directly, as this reduces the search space and speeds up learning (Furelos-Blanco, Law, Jonsson, et al. 2021, p. 1046).

EXAMPLE 10. For the RM in Figure 1, the transition $\varphi(u_0, u_1) = \text{used} \wedge \neg o \wedge \neg *$ is encoded as

$$\left\{ \begin{array}{l} \text{used_state}(u_0). \text{used_state}(u_1). \text{ed}(u_0, u_1, 1). \\ \hat{\varphi}(u_0, u_1, 1, T) :- \text{not prop}(\text{used}, T), \text{step}(T). \\ \hat{\varphi}(u_0, u_1, 1, T) :- \text{prop}(o, T), \text{step}(T). \\ \hat{\varphi}(u_0, u_1, 1, T) :- \text{prop}(*, T), \text{step}(T). \end{array} \right\}.$$

We now define the ASP encoding for *traces*, also following Furelos-Blanco, Law, Jonsson, et al. (2021).

Definition 4.2 (ASP representation of a label trace). Given a label trace $\lambda = \langle L_0, \dots, L_n \rangle$, $M(\lambda)$ denotes the set of ASP facts

$$M(\lambda) = \{ \text{prop}(l, t). \mid 0 \leq t \leq n, l \in L_t \} \cup \{ \text{step}(t). \mid 0 \leq t \leq n \} \cup \{ \text{last}(n). \},$$

where $\text{last}(n)$ denotes that n is the final timestep.

Given the RM and trace encodings (Definitions 4.1 and 4.2), we define rules for RM traversal:

$$R_{\text{st}} = \left\{ \begin{array}{l} \text{st}(\emptyset, u_0). \\ \text{st}(T+1, Y) :- \text{st}(T, X), \delta(X, Y, T). \\ \text{accept} :- \text{last}(T), \text{st}(T+1, u_A). \\ \text{reject} :- \text{last}(T), \text{st}(T+1, u_R). \end{array} \right\},$$

where $\text{st}(T, X)$ denotes that the state at timestep T is X ; $\delta(X, Y, T)$ indicates that the transition from X to Y is satisfied at timestep T ; and accept and reject respectively indicate whether the trace is accepted or rejected. The rules encode that: traversals start from the initial state; the next state is determined based on the currently satisfied transition; and that the trace is accepted (resp. rejected) if the last state is u_A (resp. u_R). The $\delta(X, Y, T)$ atoms are defined via:

$$R_{\delta} = \left\{ \begin{array}{l} \varphi(X, Y, E, T) :- \text{not } \hat{\varphi}(X, Y, E, T), \text{ed}(X, Y, E), \text{step}(T). \\ \text{out_}\varphi(X, T) :- \varphi(X, _, _, T). \\ \delta(X, Y, T) :- \varphi(X, Y, _, T). \\ \delta(X, X, T) :- \text{not out_}\varphi(X, T), \text{used_state}(X), \text{step}(T). \end{array} \right\},$$

which encode (i) φ as the negation of $\hat{\varphi}$; (ii) whether an outgoing transition from state X is satisfied at timestep T ; (iii) $\delta(X, Y, T)$ holds when the formula on an edge from X to Y is satisfied at timestep T ; and (iv) $\delta(X, X, T)$ holds (self-loop) when no outgoing transition from X is satisfied at timestep T . Note that the $\hat{\varphi}$ rules (Definition 4.1) are evaluated using the trace encoding (Definition 4.2).

4.3 Learning from Answer Sets Task

We learn RMs using ILASP (Law et al. 2015), a state-of-the-art LAS system. A LAS task consists of background knowledge B , a hypothesis space S_M , and a set of WCDPI examples E (Section 2.3). To apply LAS to noisy traces, we sample grounded traces from each noisy trace, creating training examples that represent the underlying trace while accounting for uncertainty. This allows us to extend the learning approach of Furelos-Blanco, Law, Jonsson, et al. (2021) to the noisy setting.

4.3.1 Background Knowledge. The background knowledge $B = R_{st} \cup R_\delta \cup R_U$ consists of the traversal rules from the previous subsection (R_{st} and R_δ , Section 4.2), which determine whether an RM accepts or rejects a trace, together with rules defining the state space:

$$R_U = \{\text{state}(u). \mid u \in U_{\max}\} \cup \{\text{state}(u_A). \mid u_A \in U_{\max}\} \cup \{\text{state}(u_R). \mid u_R \in U_{\max}\} \\ \cup \left\{ \begin{array}{l} \text{used_state}(U) :- \text{state}(U), \text{ed}(U, _, _). \\ \text{used_state}(U) :- \text{state}(U), \text{ed}(_, U, _). \end{array} \right\}.$$

Here $U_{\max}$ is the maximum set of states available to ILASP; hence, the learned RM may use a subset. When $|U_{\max}|$ is large, the search space contains many equivalent RMs differing only in state identifiers. To reduce this redundancy, we extend the symmetry-breaking constraints from Furelos-Blanco, Law, Jonsson, et al. (2021) by requiring that used states have the lowest possible identifiers. The set of used states is determined via $\text{used_state}(U)$ atoms indicating that state U has incoming or outgoing edges.

4.3.2 Hypothesis Space. The hypothesis space S_M defines the set of all possible RMs that can be learned. Based on the RM encoding from Section 4.2, we define:

$$S_M = \left\{ \begin{array}{l} \text{ed}(u, u', i). \\ \hat{\phi}(u, u', i, T) :- \text{prop}(l, T), \text{step}(T). \\ \hat{\phi}(u, u', i, T) :- \text{not prop}(l, T), \text{step}(T). \end{array} \mid \begin{array}{l} u \in U_{\max} \setminus \{u_A, u_R\}, \\ u' \in U_{\max} \setminus \{u\}, \\ i \in [1, \kappa], l \in \mathcal{P} \end{array} \right\},$$

where κ is the maximum number of edges between any two states.

4.3.3 Example Set. The example set E consists of WCDPIs generated from grounded traces (Sections 4.1.1 and 4.1.2). Since trace termination is deterministic (Assumption 2), we know with certainty whether each trace is a goal, dead-end, or incomplete trace. Below we describe the components of the WCDPI $\langle e_{id}, e_{pen}, \langle e_{inc}, e_{exc} \rangle, e_{ctx} \rangle$ resulting from each *unique* sampled grounded trace λ' .

Identifier. The identifier e_{id} is unique for each WCDPI.

Penalty. The penalty e_{pen} balances class frequency, trace likelihood, and trace source through three components:

- (1) **Class-based weight.** To address class imbalance, we assign a base weight inversely proportional to the trace class frequency (Huang et al. 2016): $w_c = \frac{1}{n_c}$, where n_c is the number of examples of class $c \in \{\text{goal}, \text{dead-end}, \text{incomplete}\}$.
- (2) **Deduplication count.** If k identical grounded traces exist, we retain one WCDPI with penalty multiplied by k . This reflects that frequently sampled grounded traces are more likely under the noisy labelling function and should be prioritised. We track counts separately for traces from noisy labels (k_{λ}) and from the existing RM (k_{RM}).
- (3) **Trace origin weight.** We introduce a hyperparameter $w_{t_o} \in \mathbb{R}^+$ that controls the importance of RM-derived examples. Setting $w_{t_o} = 0$ discards RM-derived examples entirely, while $w_{t_o} = 1$ weights both sources equally.

Thus, the final penalty is $e_{pen} = w_c(k_{\lambda} + w_{t_o}k_{RM})$ for a WCDPI of class c .

Inclusions and Exclusions. The inclusion-exclusion pair $\langle e_{inc}, e_{exc} \rangle$ depends on the trace type:

- $\langle \{\text{accept}\}, \{\text{reject}\} \rangle$ for *goal* traces,
- $\langle \{\text{reject}\}, \{\text{accept}\} \rangle$ for *dead-end* traces, and
- $\langle \{\}, \{\text{accept}, \text{reject}\} \rangle$ for *incomplete* traces.

These pairs encode that goal traces must be accepted and not rejected, dead-end traces must be rejected and not accepted, and incomplete traces must be neither accepted nor rejected, consistent with the RM validity definition in Section 2.2.

Context. The context e_{ctx} consists of the ASP representation $M(\lambda')$ of the grounded trace. Additionally, to speed up learning, we include a constraint ensuring the terminal state is reached after the final label:

$$:- \text{last}(T), T' \leq T, \text{st}(T', u_A). \quad (\text{goal traces})$$

$$:- \text{last}(T), T' \leq T, \text{st}(T', u_R). \quad (\text{dead-end traces})$$

4.3.4 Scoring and Complexity Control. ILASP scores hypotheses by balancing rule length and unsatisfied example penalties (see Section 2.3.2). We control this trade-off through two mechanisms:

- (1) **Penalty budget.** Adjusting the total penalty sum $\sum_{e \in E} e_{pen}$ controls example coverage. A higher budget requires the learned RM to satisfy more examples, potentially increasing complexity.
- (2) **Length function.** Customising the rule length function penalises complex RMs. We adjust the weight given to hypothesis length relative to the unsatisfied penalty.

This allows us to find RMs that balance simplicity (fewer states, simpler formulas) with example coverage. To simplify the learning task, we additionally discard any WCDPIs whose penalty falls below a threshold t_{epen} .

5 Exploiting Reward Machines from Noisy Traces

In this section, we describe how PROB-IRM exploits the structure of an RM when labels are noisy. Akin to Li, Z. Chen, Klassen, et al. (2024), PROB-IRM learns policies with PPO (Schulman et al. 2017) using a neural network that takes as input the current environment state and the RM state belief, and outputs both the action distribution and value function.

The key challenge is handling uncertainty in the RM state: since labels are noisy, the agent maintains a belief distribution over RM states rather than a single deterministic state. We introduce the RM state belief in Section 5.1, extend the reward function to beliefs in Section 5.2, and describe probabilistic reward shaping in Section 5.3.

5.1 Reward Machine State Belief

Due to the noisy labelling function, the current RM state cannot be known with certainty; therefore, we determine the *RM state belief* (Li, Z. Chen, Klassen, et al. 2024; Li, Z. Chen, Vaezipoor, et al. 2022).

To compute the belief, we need the probability of observing a specific label $L \in 2^P$ at a transition. Under Assumption 1, let $[\tilde{\mathcal{L}}(s_t, a_t, s_{t+1})]_l$ denote the probability assigned by the noisy labelling function to proposition l at transition $\langle s_t, a_t, s_{t+1} \rangle$. The label probability is:

$$P(L \mid s_t, a_t, s_{t+1}) = \prod_{l \in P} p_l,$$

where

$$p_l = \begin{cases} [\tilde{\mathcal{L}}(s_t, a_t, s_{t+1})]_l & \text{if } l \in L; \\ 1 - [\tilde{\mathcal{L}}(s_t, a_t, s_{t+1})]_l & \text{if } l \notin L. \end{cases}$$

Definition 5.1 (RM state belief). The RM state belief $\tilde{u}_t \in \Delta(U)$ is a probability distribution over RM states at timestep t . The initial belief is:

$$\tilde{u}_0(u) = \sum_{L_0 \in 2^P} P(L_0 \mid \emptyset, \emptyset, s_0) \mathbf{1}[\delta^u(u_0, L_0) = u]. \quad (2)$$

Belief updates are performed as follows:

$$\tilde{u}_{t+1}(u) = \sum_{\substack{u_t \in U, \\ L_t \in 2^P}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u], \quad (3)$$

where δ^u is the RM state-transition function and $\mathbf{1}$ is the indicator function.

EXAMPLE 11. Consider the RM from Figure 1 and the noisy trace $\tilde{\lambda} = \langle \{\text{⚡} : 0.0002, o : 0.0001\}, \{\text{⚡} : 0.6513, o : 0.0001\} \rangle$ resulting from the history $h = \langle \langle 4, 6 \rangle, \text{left}, \langle 3, 6 \rangle \rangle$ in Example 6. We compute the belief evolution using Definition 5.1, simplifying summations to include only non-zero terms:

- Step $t = 0$. The agent starts in environment state $s_0 = \langle 4, 6 \rangle$. We compute the initial belief using the noisy label $\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0) = \{\text{⚡} : 0.0002, o : 0.0001\}$:

$$\begin{aligned} \tilde{u}_0(u_0) &= \sum_{\substack{L_0 \models \text{⚡} \wedge \neg o \wedge \neg *, \\ L_0 \models \text{⚡} \wedge o \wedge \neg *, \\ L_0 \models *}} P(L_0 \mid \emptyset, \emptyset, s_0) \mathbf{1}[\delta^u(u_0, L_0) = u_0] \\ &= P(\emptyset \mid \emptyset, \emptyset, s_0) + P(\{o\} \mid \emptyset, \emptyset, s_0) \\ &= \left[(1 - [\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0)]_{\text{⚡}})(1 - [\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0)]_o) \right] \\ &\quad + \left[(1 - [\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0)]_{\text{⚡}})[\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0)]_o \right] \\ &= [(1 - 0.0002)(1 - 0.0001)] + [(1 - 0.0002)(0.0001)] \\ &= 0.9998, \end{aligned}$$

$$\begin{aligned} \tilde{u}_0(u_1) &= \sum_{L_0 \models \text{⚡} \wedge \neg o \wedge \neg *} P(L_0 \mid \emptyset, \emptyset, s_0) \mathbf{1}[\delta^u(u_0, L_0) = u_1] \\ &= P(\{\text{⚡}\} \mid \emptyset, \emptyset, s_0) \\ &= [\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0)]_{\text{⚡}} (1 - [\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0)]_o) \\ &= 0.0002(1 - 0.0001) \\ &\approx 0.0002, \end{aligned}$$

$$\begin{aligned} \tilde{u}_0(u_A) &= \sum_{L_0 \models \text{⚡} \wedge o \wedge \neg *} P(L_0 \mid \emptyset, \emptyset, s_0) \mathbf{1}[\delta^u(u_0, L_0) = u_A] \\ &= P(\{\text{⚡}, o\} \mid \emptyset, \emptyset, s_0) \\ &= [\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0)]_{\text{⚡}} [\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0)]_o \\ &= 0.0002 \cdot 0.0001 \\ &\approx 0.0000, \end{aligned}$$

$$\begin{aligned} \tilde{u}_0(u_R) &= \sum_{L_0 \models *} P(L_0 \mid \emptyset, \emptyset, s_0) \mathbf{1}[\delta^u(u_0, L_0) = u_R] \\ &= 0.0 \quad (\text{since } [\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0)]_* = 0). \end{aligned}$$

- Step $t = 1$. After the transition $\langle s_0, a_0, s_1 \rangle$, where $s_0 = \langle 4, 6 \rangle$, $a_0 = \text{left}$, $s_1 = \langle 3, 6 \rangle$, and $\tilde{\mathcal{L}}(s_0, a_0, s_1) = \{\text{⚡} : 0.6513, o : 0.0001\}$:

$$\begin{aligned} \tilde{u}_1(u_0) &= \sum_{\substack{L_1 \models \text{⚡} \wedge \neg o \wedge \neg *, \\ L_1 \models \text{⚡} \wedge o \wedge \neg *, \\ L_1 \models *}} P(L_1 \mid s_0, a_0, s_1) \tilde{u}_0(u_0) \\ &= \tilde{u}_0(u_0) [P(\emptyset \mid s_0, a_0, s_1) + P(\{o\} \mid s_0, a_0, s_1)] \end{aligned}$$

$$\begin{aligned}
&= \tilde{u}_0(u_0) \left[(1 - [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_{\text{⬇}})(1 - [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_o) \right. \\
&\quad \left. + (1 - [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_{\text{⬇}})[\tilde{\mathcal{L}}(s_0, a_0, s_1)]_o \right] \\
&= 0.9998 [(1 - 0.6513)(1 - 0.0001) + (1 - 0.6513)0.0001] \\
&\approx 0.3486,
\end{aligned}$$

$$\begin{aligned}
\tilde{u}_1(u_1) &= \sum_{L_1 \models \text{⬇} \wedge \neg o \wedge \neg *} P(L_1 \mid s_0, a_0, s_1) \tilde{u}_0(u_0) + \sum_{\substack{L_1 \models o \wedge \neg *, \\ L_1 \not\models *}} P(L_1 \mid s_0, a_0, s_1) \tilde{u}_0(u_1) \\
&= \tilde{u}_0(u_0) P(\{\text{⬇}\} \mid s_0, a_0, s_1) + \tilde{u}_0(u_1) [P(\emptyset \mid s_0, a_0, s_1) + P(\{\text{⬇}\} \mid s_0, a_0, s_1)] \\
&= \tilde{u}_0(u_0) \left[[\tilde{\mathcal{L}}(s_0, a_0, s_1)]_{\text{⬇}}(1 - [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_o) \right] \\
&\quad + \tilde{u}_0(u_1) \left[(1 - [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_{\text{⬇}})(1 - [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_o) \right. \\
&\quad \left. + [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_{\text{⬇}}(1 - [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_o) \right] \\
&= 0.9998 [0.6513(1 - 0.0001)] \\
&\quad + 0.0002 [(1 - 0.6513)(1 - 0.0001) + 0.6513(1 - 0.0001)] \\
&\approx 0.6513,
\end{aligned}$$

$$\begin{aligned}
\tilde{u}_1(u_A) &= \sum_{L_1 \models \text{⬇} \wedge o \wedge \neg *} P(L_1 \mid s_0, a_0, s_1) \tilde{u}_0(u_0) + \sum_{L_1 \models o \wedge \neg *} P(L_1 \mid s_0, a_0, s_1) \tilde{u}_0(u_1) \\
&= \tilde{u}_0(u_0) P(\{\text{⬇}, o\} \mid s_0, a_0, s_1) + \tilde{u}_0(u_1) [P(\{o\} \mid s_0, a_0, s_1) + P(\{\text{⬇}, o\} \mid s_0, a_0, s_1)] \\
&= \tilde{u}_0(u_0) \left[[\tilde{\mathcal{L}}(s_0, a_0, s_1)]_{\text{⬇}} [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_o \right] \\
&\quad + \tilde{u}_0(u_1) \left[(1 - [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_{\text{⬇}}) [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_o \right. \\
&\quad \left. + [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_{\text{⬇}} [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_o \right] \\
&= 0.9998 [0.6513 \cdot 0.0001] + 0.0002 [(1 - 0.6513) \cdot 0.0001 + 0.6513 \cdot 0.0001] \\
&\approx 0.0001,
\end{aligned}$$

$$\begin{aligned}
\tilde{u}_1(u_R) &= \sum_{L_1 \models *} P(L_1 \mid s_0, a_0, s_1) \tilde{u}_0(u_0) + \sum_{L_1 \models *} P(L_1 \mid s_0, a_0, s_1) \tilde{u}_0(u_1) \\
&= 0.0 \quad (\text{since } [\tilde{\mathcal{L}}(s_0, a_0, s_1)]_* = 0).
\end{aligned}$$

In summary, the resulting beliefs are:

$$\begin{aligned}
\tilde{u}_0 &\approx \begin{bmatrix} 0.9998 & 0.0002 & 0.0000 & 0.0000 \end{bmatrix}^\top, \\
\tilde{u}_1 &\approx \begin{bmatrix} 0.3486 & 0.6513 & 0.0001 & 0.0000 \end{bmatrix}^\top.
\end{aligned}$$

The belief evolution shows the agent's increasing certainty about having observed ⬇ : starting with near certainty of being in u_0 at $t = 0$, the belief shifts primarily to u_1 by $t = 1$, reflecting the strong ⬇ detection upon moving left. However, substantial probability mass remains on u_0 , reflecting uncertainty due to the noisy labels.

5.2 Reward over Reward Machine State Beliefs

In the deterministic case, the agent receives reward $\delta^r(u_t, u_{t+1})$ when transitioning between RM states. However, with noisy labels, the agent maintains a belief distribution over RM states rather than knowing the current state with certainty. We extend the reward-transition function to beliefs by computing the expected reward over all possible state transitions weighted by their probabilities.

Definition 5.2 (Reward over RM state beliefs). Given a reward machine $\mathcal{RM}$, a pair of state beliefs $\tilde{u}_t, \tilde{u}_{t+1} \in \Delta(U)$, and a transition $\langle s_t, a_t, s_{t+1} \rangle \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, the reward-transition over beliefs $\tilde{\delta}^r : \Delta(U) \times \Delta(U) \rightarrow \mathbb{R}$ is

$$\tilde{\delta}^r(\tilde{u}_t, \tilde{u}_{t+1}) = \sum_{\substack{u_t, u_{t+1} \in U, \\ L_t \in 2^P}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u_{t+1}] \delta^r(u_t, u_{t+1}). \quad (4)$$

While Equation (4) holds for any reward-transition function, under Assumption 3, it simplifies to a closed form that depends only on changes in belief at the terminal states:

PROPOSITION 5.3. *Under Assumption 3, the reward-transition over beliefs is:*

$$\tilde{\delta}^r(\tilde{u}_t, \tilde{u}_{t+1}) = (\tilde{u}_{t+1}(u_A) - \tilde{u}_t(u_A)) - (\tilde{u}_{t+1}(u_R) - \tilde{u}_t(u_R)). \quad (5)$$

PROOF. See Appendix B. □

5.3 Probabilistic Reward Shaping

Reward shaping provides intermediate rewards to guide agent learning. *Potential-based reward shaping* (Ng et al. 1999) defines these rewards by evaluating a *potential function* $\Phi : \mathcal{S} \rightarrow \mathbb{R}$ at consecutive states and taking their difference. Formally, the shaped reward is

$$r_s(s_t, s_{t+1}) = \gamma \Phi(s_{t+1}) - \Phi(s_t),$$

where γ is the MDP's discount factor. Crucially, this approach preserves the set of optimal policies.

In the context of reward machines, several works (Camacho, O. Chen, et al. 2017; Camacho, Toro Icarte, et al. 2019; Furelos-Blanco, Law, Jonsson, et al. 2021) leverage the RM structure for reward shaping by defining the potential function over RM states, $\Phi : U \rightarrow \mathbb{R}$, yielding the shaped reward

$$r_s(u_t, u_{t+1}) = \gamma \Phi(u_{t+1}) - \Phi(u_t).$$

In our setting with noisy labels, the agent cannot directly observe the true RM state and instead maintains a belief distribution $\tilde{u}_t \in \Delta(U)$ over RM states. We therefore extend the potential function to operate over beliefs: $\tilde{\Phi} : \Delta(U) \rightarrow \mathbb{R}$. We define $\tilde{\Phi}$ as the expected potential under the belief distribution:

$$\tilde{\Phi}(\tilde{u}_t) = \sum_{u \in U} \tilde{u}_t(u) \Phi(u).$$

The shaped reward then follows the same potential-based form:

$$\begin{aligned} r_s(\tilde{u}_t, \tilde{u}_{t+1}) &= \gamma \tilde{\Phi}(\tilde{u}_{t+1}) - \tilde{\Phi}(\tilde{u}_t) \\ &= \sum_{u \in U} (\gamma \tilde{u}_{t+1}(u) - \tilde{u}_t(u)) \Phi(u). \end{aligned}$$

This approach is analogous to potential-based shaping over state beliefs in partially observable MDPs (Eck et al. 2013).

To reward progress toward the accepting state, we define the potential function following Camacho, O. Chen, et al. (2017) and Furelos-Blanco, Law, Jonsson, et al. (2021):

$$\Phi(u) = |U| - d_{\min}(u, u_A),$$

where $d_{\min}(u, u_A)$ is the shortest path length from u to u_A . This assigns higher potential to states closer to u_A , thereby rewarding transitions that reduce the distance to the goal. If u_A is unreachable from u , we set $d_{\min}(u, u_A) = \infty$ (in practice, a sufficiently large number).

Following standard practice in reward shaping, we augment the reward over RM state beliefs $\tilde{\delta}^r$ (see Section 5.2) with the shaped reward:

$$\tilde{\delta}^{\text{RS}}(\tilde{u}_t, \tilde{u}_{t+1}) = \tilde{\delta}^r(\tilde{u}_t, \tilde{u}_{t+1}) + r_s(\tilde{u}_t, \tilde{u}_{t+1}).$$

EXAMPLE 12. Consider the reward machine in Figure 1 with consecutive RM state beliefs $\tilde{u}_t = [1 \ 0 \ 0 \ 0]^\top$ and $\tilde{u}_{t+1} = [0 \ 0.5 \ 0.5 \ 0]^\top$ (ordered as u_0, u_1, u_A, u_R), representing a transition from certainty in u_0 to equal uncertainty between u_1 and u_A . First, we compute the potential function Φ :

$$\begin{aligned} \Phi(u_0) &= |U| - d_{\min}(u_0, u_A) = 4 - 1 = 3, & \Phi(u_1) &= |U| - d_{\min}(u_1, u_A) = 4 - 1 = 3, \\ \Phi(u_A) &= |U| - d_{\min}(u_A, u_A) = 4 - 0 = 4, & \Phi(u_R) &= |U| - d_{\min}(u_R, u_A) = 4 - \infty = -\infty. \end{aligned}$$

With discount factor $\gamma = 0.9$, the shaped reward is:

$$\begin{aligned} r_s(\tilde{u}_t, \tilde{u}_{t+1}) &= \sum_{u \in U} (\gamma \tilde{u}_{t+1}(u) - \tilde{u}_t(u)) \Phi(u) \\ &= (\gamma \tilde{u}_{t+1}(u_0) - \tilde{u}_t(u_0)) \Phi(u_0) + (\gamma \tilde{u}_{t+1}(u_1) - \tilde{u}_t(u_1)) \Phi(u_1) \\ &\quad + (\gamma \tilde{u}_{t+1}(u_A) - \tilde{u}_t(u_A)) \Phi(u_A) + (\gamma \tilde{u}_{t+1}(u_R) - \tilde{u}_t(u_R)) \Phi(u_R) \\ &= (0.9 \cdot 0 - 1) \cdot 3 + (0.9 \cdot 0.5 - 0) \cdot 3 \\ &\quad + (0.9 \cdot 0.5 - 0) \cdot 4 + (0.9 \cdot 0 - 0) \cdot (-\infty) \\ &= 0.15. \end{aligned}$$

The positive shaped reward reflects the increased probability of being in u_A .

6 Interleaving Exploitation and Learning of Reward Machines from Noisy Traces

In the noiseless setting, prior work (Furelos-Blanco, Law, Jonsson, et al. 2021; M. Hasanbeig, Jeppu, et al. 2021; Toro Icarte, Waldie, et al. 2019; Xu et al. 2020) triggers RM relearning when the current candidate RM fails to explain an observed trace. For instance, if the agent successfully completes the task but the corresponding trace does not reach the accepting state, this counterexample indicates the RM must be revised. With noisy labels, however, such a counterexample-based approach becomes unreliable: we cannot determine with certainty whether a trace genuinely contradicts the RM. Moreover, attempting to explain every observed noisy trace would cause the learned RM to overfit to sensor errors.

Figure 4 provides an overview of PROB-IRM, our approach for learning RMs from noisy labels. To address the challenges introduced by noisy observations, we propose a criterion for determining when to relearn the RM. We describe this criterion in Section 6.1 and present the complete algorithm in Section 6.2.

6.1 Reward Machine Relearning Criterion

In the noiseless setting, prior work triggers relearning when the RM is not valid with respect to a trace; namely, given a noiseless trace λ and the traversal $\mathcal{RM}(\lambda) = \langle v_0, \dots, v_{n+1} \rangle$, a new RM is learned if (i) λ is a goal trace and $v_{n+1} \neq u_A$; (ii) λ is a dead-end trace and $v_{n+1} \neq u_R$; or (iii) λ is an incomplete trace and $v_{n+1} \in \{u_A, u_R\}$. However, with noisy labels, we cannot accurately determine the last RM state of a traversal and thus cannot directly compare it to the trace category to detect errors.

We propose a criterion for RM relearning that builds upon RM state beliefs (Section 5.1) and deterministic termination information (Assumption 2). The criterion is defined using an error metric that measures how well the RM state belief at termination aligns with the ground-truth termination information using cross-entropy.

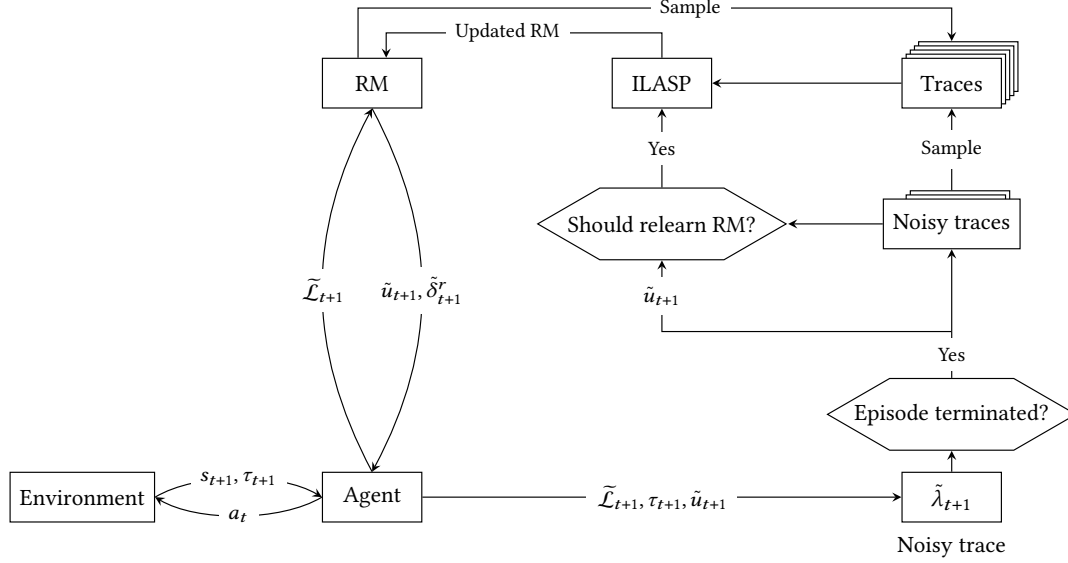


Fig. 4. Overview of the PROB-IRM algorithm that interleaves RM learning and exploitation from noisy traces. At each timestep, the agent takes action a_t and observes the next state s_{t+1} and termination information τ_{t+1} . The agent computes the noisy label $\tilde{L}_{t+1}$ from the transition $\langle s_t, a_t, s_{t+1} \rangle$. The noisy label and current RM are used to determine the RM state belief $\tilde{u}_{t+1}$ (Section 5.1) and RM reward $\tilde{\delta}_{t+1}$ (Section 5.2). The episode trace $\tilde{\lambda}_{t+1}$ is updated with the noisy label. At episode termination, the trace is stored and we determine whether to relearn the RM using the final RM state belief (Section 6.1). If relearning is triggered, grounded traces are sampled from the noisy traces (Section 4.1.1) and from the current RM (Section 4.1.2). These traces are provided to ILASP to learn a new RM.

Definition 6.1 (RM error). Given a noisy trace $\tilde{\lambda} = \langle L_0, \dots, L_n \rangle$, the resulting RM state belief $\tilde{u}_n \in \Delta(U)$, and termination information $\tau_n = \langle \tau_n^T, \tau_n^G \rangle$ at the final step n , the RM error is

$$\text{error}(v, \tilde{v}) = \text{CROSSENTROPY}(v, \tilde{v}) = -v^\top \log(\tilde{v}),$$

where $v \in \{0, 1\}^3$ is the ground-truth one-hot vector encoding the trace category (goal, dead-end, and incomplete) based on termination information, and $\tilde{v} \in [0, 1]^3$ encodes the RM state belief over terminal (u_A and u_R) and non-terminal states. Formally,

$$v = \begin{bmatrix} \mathbf{1}[\tau_n^T = \top \wedge \tau_n^G = \top] \\ \mathbf{1}[\tau_n^T = \top \wedge \tau_n^G = \perp] \\ \mathbf{1}[\tau_n^T = \perp] \end{bmatrix}, \quad \tilde{v} = \begin{bmatrix} \tilde{u}_n(u_A) \\ \tilde{u}_n(u_R) \\ 1 - \tilde{u}_n(u_A) - \tilde{u}_n(u_R) \end{bmatrix}.$$

EXAMPLE 13. Consider the RM state belief $\tilde{u}_1 = [0.3486 \quad 0.6513 \quad 0.0001 \quad 0.0000]^\top$ from Example 11 and the termination information $\tau_1 = \langle \perp, \perp \rangle$ (incomplete trace). The vectors are:

$$v = [0 \quad 0 \quad 1]^\top, \quad \tilde{v} = [0.0001 \quad 0.0000 \quad 0.9999]^\top.$$

The RM error is $-v^\top \log(\tilde{v}) = -\log(0.9999) \approx 0.0$, indicating that the RM state belief accurately estimates that a terminal state has not been reached.

We trigger RM relearning based on the cumulative RM error since the last update. After each episode termination, we compute the RM error for that episode and add it to a running sum. When the average error across episodes

(total error divided by the number of episodes since the last relearning) exceeds a threshold β , we trigger relearning and reset the cumulative error to zero.

To reduce sensitivity to the choice of β , we introduce two techniques:

- (1) **Warm-up period.** After each relearning, we require a minimum number of episodes before the RM can be updated again, preventing premature relearning.
- (2) **Score-based gating.** We only update the RM if the newly learned hypothesis achieves a better ILASP score (Definition 2.3) than the current RM, preventing updates that do not improve example fit.

Together, these techniques substantially reduce sensitivity to β , as demonstrated in Section 7.4.6. The warm-up period ensures training progress even with a conservatively low threshold, while score-based gating prevents unnecessary updates when an error spike is due to temporary exploration rather than RM inadequacy.

6.2 Interleaved Learning Algorithm

We now describe PROB-IRM, our method for interleaving RM learning from noisy traces with RL. Algorithm 1 presents the main interleaving loop, and Algorithm 2 describes the `RMLEARNER` class that manages RM updates.

Algorithm 1 initialises the RM, policy, and RM learner (lines 1–3). The initial RM simply loops on the initial state and thus does not accept or reject any traces. The algorithm then runs for a fixed number of iterations. In each iteration, multiple episode trajectories are sampled using the `SAMPLETRAJECTORY` function (lines 5–7) and used to update the policy (line 8). If the relearning criterion is met, a new RM is learned and the policy is reinitialised (lines 9–11).

Unlike prior work (Furelos-Blanco, Law, Jonsson, et al. 2021; Xu et al. 2020) that uses iterative deepening to find a minimal RM by incrementally increasing the state count until all examples are covered, we use a fixed maximum state set $U_{\max}$ for all RM learning iterations. Each learned RM may use only a subset of $U_{\max}$, but the maximum remains constant. This is necessary because examples carry noise penalties: iterative deepening would terminate prematurely by ignoring noisy examples rather than increasing the state count. This strategy is also used in other RM learning work with non-noisy traces (M. Hasanbeig, Jeppu, et al. 2021; Toro Icarte, Waldie, et al. 2019).

The `SAMPLETRAJECTORY` function (lines 12–30) runs an episode, collecting a trajectory for policy learning and a noisy trace with its RM error for RM learning. During an episode, the agent selects an action based on the current environment state and RM state belief, executes it in the environment (lines 20–21), and updates the RM state belief, noisy trace, and trajectory (lines 22–24). The episode terminates when: (i) the environment signals termination; (ii) the RM state belief is concentrated on terminal states; or (iii) the maximum episode length is reached. At the end of the episode, the agent updates the RM learning examples using the noisy trace and current RM (line 28) and increments the accumulated RM error used to determine whether to learn a new RM (line 29).

The `RMLEARNER` class (Algorithm 2) implements the relearning criterion (Section 6.1) and ILASP-based RM learning (Section 4). The constructor (lines 1–4) initialises an empty example set, along with an episode count and RM error accumulator that track state since the last RM update. The `SHOULDRELEARN` method (lines 5–8) implements the relearning criterion by checking whether the average RM error exceeds the threshold. The `RELEARNRM` method (lines 9–12) resets the episode count and RM error accumulator, then calls ILASP to learn a new RM using the current example set. The `UPDATEEXAMPLES` method (lines 13–16) updates the example set from both noisy traces and the previous RM (Section 4.1). Finally, `UPDATERMERROR` (lines 17–22) computes the RM error (Section 6.1) and updates the episode count and error accumulator.

7 Experimental Results

To evaluate PROB-IRM, we design experiments across discrete and continuous domains under varying noise conditions. Our experiments address the following research questions:

Algorithm 1 PROB-IRM algorithm

Require: $\mathcal{P}$ (propositions), $U_{\max}$ (RM states), κ (max. edges between states), β (relearning threshold),
warmup_episodes (episodes before relearning), max_ep_len (max. episode length),
num_iters (iterations), num_episodes_per_iter (episodes per iteration)

```

1:  $\mathcal{RM} \leftarrow \text{INITRM}(U_{\max})$ 
2:  $\pi \leftarrow \text{INITPOLICY}(\mathcal{RM})$ 
3:  $\text{rm\_learner} \leftarrow \text{RMLERNER}()$ 
4: for  $i \leftarrow 1, \dots, \text{num\_iters}$  do
5:   trajectories  $\leftarrow \{\}$ 
6:   for  $ep \leftarrow 1, \dots, \text{num\_episodes\_per\_iter}$  do
7:     trajectories  $\leftarrow \text{trajectories} \oplus \text{SAMPLETRAJECTORY}(\mathcal{RM}, \pi, \text{rm\_learner}, \text{max\_ep\_len})$ 
8:      $\text{UPDATEPOLICY}(\pi, \text{trajectories})$  ▷ See Section 5
9:     if  $\text{rm\_learner.SHOULORELEARN}(\beta, \text{warmup\_episodes})$  then
10:       $\mathcal{RM} \leftarrow \text{rm\_learner.RELEARNRM}(\mathcal{P}, U_{\max}, \kappa)$ 
11:       $\pi \leftarrow \text{INITPOLICY}(\mathcal{RM})$ 

12: function  $\text{SAMPLETRAJECTORY}(\mathcal{RM}, \pi, \text{rm\_learner}, \text{max\_ep\_len})$ 
13:    $s \leftarrow \text{ENVINITIALSTATE}()$ 
14:    $\tilde{u} \leftarrow \text{INITBELIEF}(\mathcal{RM}, \tilde{\mathcal{L}}(\emptyset, \emptyset, s))$  ▷ Using Equation (2)
15:    $t \leftarrow 0$  ▷ Initialise episode step counter
16:    $\tilde{\lambda} \leftarrow \langle \rangle$  ▷ Initialise noisy trace
17:    $\mathcal{T} \leftarrow \langle \rangle$  ▷ Initialise raw trajectory
18:    $\text{done} \leftarrow \perp$  ▷ Initialise episode termination indicator
19:   while  $\text{done} = \perp$  do
20:      $a \leftarrow \pi(s, \tilde{u})$ 
21:      $s', \tau \leftarrow \text{ENVSTEP}(s, a)$ 
22:      $\tilde{u}' \leftarrow \text{UPDATEBELIEF}(\mathcal{RM}, \tilde{\mathcal{L}}(s, a, s'), \tilde{u})$  ▷ Using Equation (3)
23:      $\tilde{\lambda} \leftarrow \tilde{\lambda} \oplus \tilde{\mathcal{L}}(s, a, s')$ 
24:      $\mathcal{T} \leftarrow \mathcal{T} \oplus \langle s, a, s', \delta^r(\tilde{u}, \tilde{u}'), \tilde{u}' \rangle$ 
25:      $s \leftarrow s'; \tilde{u} \leftarrow \tilde{u}'; t \leftarrow t + 1$ 
26:      $\text{is\_terminal\_rm\_state} \leftarrow (\tilde{u}(u_A) + \tilde{u}(u_R)) \geq 0.5$ 
27:      $\text{done} \leftarrow \tau^T = \top$  or  $\text{is\_terminal\_rm\_state}$  or  $t > \text{max\_ep\_len}$ 
28:      $\text{rm\_learner.UPDATEEXAMPLES}(\tilde{\lambda}, \mathcal{RM}, \tau)$ 
29:      $\text{rm\_learner.UPDATERMERROR}(\tau, \tilde{u})$ 
30:   return  $\mathcal{T}$ 

```

RQ1 Does PROB-IRM enable agents to solve non-Markovian reward tasks under noisy labels?

RQ2 Is learning an RM preferable to using recurrent policies?

RQ3 Does noise-aware RM learning outperform noise-unaware alternatives?

RQ4 How does PROB-IRM compare to using handcrafted RMs?

RQ5 How sensitive is PROB-IRM to different sensor noise levels?

RQ6 Does PROB-IRM scale to continuous domains?

Algorithm 2 The RMLEARNER class

```

1: function RMLEARNER()
2:   self.E  $\leftarrow$  {} ▷ Set of examples
3:   self.num_episodes  $\leftarrow$  0 ▷ Episodes since last RM learning
4:   self.rm_error_sum  $\leftarrow$  0 ▷ RM error sum since last RM learning
5:   function SHOULDRELEARN( $\beta$ , warmup_episodes)
6:     if self.num_episodes < warmup_episodes then
7:       return  $\perp$ 
8:     return self.rm_error_sum/self.num_episodes >  $\beta$ 
9:   function RELEARNRM( $\mathcal{P}$ ,  $U_{\max}$ ,  $\kappa$ )
10:    self.num_episodes  $\leftarrow$  0
11:    self.rm_error_sum  $\leftarrow$  0
12:    return ILASP.LEARNRM(self.E,  $\mathcal{P}$ ,  $U_{\max}$ ,  $\kappa$ ) ▷ See Section 4.3
13:   function UPDATEEXAMPLES( $\tilde{\lambda}$ ,  $\mathcal{RM}$ ,  $\tau$ ) ▷ See Section 4.1
14:      $E_{\tilde{\lambda}} \leftarrow$  GENERATEEXAMPLES( $\tilde{\lambda}$ ,  $\tau$ )
15:      $E_{\mathcal{RM}} \leftarrow$  GENERATERMEXAMPLES( $\mathcal{RM}$ )
16:     self.E  $\leftarrow$  self.E  $\cup$   $E_{\tilde{\lambda}} \cup E_{\mathcal{RM}}$ 
17:   function UPDATERMERROR( $\tau$ ,  $\tilde{u}$ ) ▷ See Section 6.1
18:      $v \leftarrow [1[\tau^T = \top \wedge \tau_n^G = \top] \quad 1[\tau^T = \top \wedge \tau_n^G = \perp] \quad 1[\tau^T = \perp]]^\top$ 
19:      $\tilde{v} \leftarrow [\tilde{u}(u_A) \quad \tilde{u}(u_R) \quad 1 - \tilde{u}(u_A) - \tilde{u}(u_R)]^\top$ 
20:     rm_error  $\leftarrow$  CROSSENTROPY( $v$ ,  $\tilde{v}$ )
21:     self.num_episodes  $\leftarrow$  self.num_episodes + 1
22:     self.rm_error_sum  $\leftarrow$  self.rm_error_sum + rm_error

```

We structure this section as follows. Section 7.1 describes our experimental setup, including domains, baselines, sensor configurations, and hyperparameters. Sections 7.2 and 7.3 present results addressing the research questions above. Section 7.4 analyses key design choices through ablation studies, covering reward shaping, belief updating, relearning criteria, example generation, and class rebalancing. Finally, Section 7.5 examines computational overhead.

Source code is available at <https://github.com/rparac/Prob-IRM>, building on the RLlib framework (Liang et al. 2018). The raw, unaggregated data of all experiments (per-run learning curves, learned RMs, and ILASP logs) is available at <https://github.com/rparac/Prob-IRM/releases/tag/jair-data-v1>.

7.1 Experimental Setup

We describe our experimental setup, covering policy learning, domains, sensor configurations, hyperparameters, and performance metrics. Experiments run on a CPU cluster with nodes containing Intel Xeon Platinum 8358 processors (32 cores, 2.60 GHz) and 256 GB RAM.

7.1.1 Policy Learning and Baselines. We use PPO (Schulman et al. 2017) to train agent policies. Unlike the tabular QRM algorithm from our prior work (Parac et al. 2024), PPO handles continuous state spaces and avoids discretising RM state beliefs. While the choice of RL algorithm is orthogonal to our method, we select PPO for its strong empirical performance and recent use in RM literature (Ardon, Furelos-Blanco, Parać, et al. 2025; Li, Klassen, et al. 2025; Li, Z. Chen, Klassen, et al. 2024).

We compare four approaches:

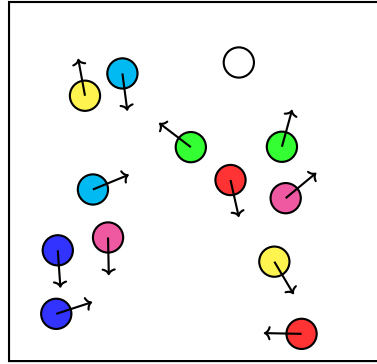


Fig. 5. Visualisation of the WATERWORLD domain (Toro Icarte, Klassen, R. A. Valenzano, et al. 2018). The agent is depicted by the white ball.

- **PROB-IRM** (ours): The network receives the environment observation and RM state belief vector (Section 5.1) as input. Learns RMs from noisy traces.
- **ISA** (Furelos-Blanco, Law, Jonsson, et al. 2021): A noise-unaware RM learning method. The network receives the environment observation and a one-hot vector indicating the current RM state (computed directly from noisy sensor output).
- **Perfect RM**: The network receives an environment observation and an RM state belief vector. Unlike PROB-IRM and ISA, it uses a perfect handcrafted RM rather than learning one. Serves as an upper bound for PROB-IRM.
- **RECURRENT PPO**: Uses an LSTM to capture history instead of an RM. Like RM-learning approaches, has no prior task structure knowledge. Receives the noisy labelling function output concatenated with an environment observation.

Network architecture details are in Section 7.1.4.

7.1.2 Domains. We evaluate PROB-IRM in the OFFICEWORLD (Toro Icarte, Klassen, R. A. Valenzano, et al. 2018) and WATERWORLD (Karpathy 2015; Toro Icarte, Klassen, R. A. Valenzano, et al. 2018) domains.

OFFICEWORLD. This grid-based navigation domain (introduced in Section 2) features deterministic movement in four directions (up, down, left, right) unless blocked by walls. The agent observes its position in the grid. We evaluate three tasks:

- **COFFEE**: collect coffee (☕) and deliver it to the office (o);
- **COFFEEMAIL**: collect coffee and mail (✉) in any order, then reach the office (o);
- **VISITABCD**: visit cells A, B, C, D in sequence.

Task difficulty varies with map layout. We run each experiment on five sets of ten randomly generated maps with randomised initial agent and object locations, reporting average performance.

WATERWORLD. This continuous state-space domain (Figure 5) features a ball-like agent floating among twelve moving coloured balls (two of each of six colours). All balls travel at constant speed. The agent applies force increments in four directions (up, down, left, right) or no force (noop). Observations comprise the agent’s absolute position and velocity, plus relative positions and velocities of all other balls. The agent observes a ball’s colour upon contact; the proposition set consists of the six colours: red, green, blue, magenta, cyan, and yellow. We run each experiment with five random seeds for initial ball positions and velocities. We evaluate three tasks:

- RGB: touch red, green, and blue balls in sequence;
- RGBC: touch red, green, blue, then cyan balls in sequence;
- RG-B: touch red, green, and blue balls in any order, with green always after red. The blue ball can be observed before, between, or after the red and green balls.

7.1.3 Sensor Configurations. We experiment with multiple sensor configurations defined by:

- **Noise targets.** Determines which sensors are noisy: *noise-first* applies noise only to the sensor for the first required proposition, while *noise-all* applies noise to all sensors. For COFFEEMAIL, which has no single first proposition, we apply noise to both ☕ and ☒ sensors in the noise-first configuration.
- **Noise level.** Characterised by sensor accuracy (the probability of correct detection). We test three levels: 1.0 (perfect), 0.9, and 0.8 (20% error rate). We set sensor sensitivity and specificity to the same value to achieve these posterior probabilities, which we refer to as *sensor confidence*.

While noiseless configurations (i.e., with sensor confidence set to 1.0) are solvable by methods assuming perfect labels, our implementation always uses the noisy framework, treating the noiseless setting as a special case.

7.1.4 Hyperparameters. We report hyperparameters in Table 1. We obtained these values through uniform random search on COFFEE and keep them constant across all experiments. Probabilistic reward shaping is enabled by default for RM-based methods, supported by the ablation study in Section 7.4.1.

7.1.5 Performance Metrics. To assess agent performance, we record the undiscounted return collected throughout each episode, excluding reward shaping when used. We aggregate returns over all training runs within each experiment to compute mean and standard deviation.

7.2 OFFICEWORLD Main Results

We evaluate PROB-IRM and the baselines for the COFFEE, COFFEEMAIL, and VISITABCD tasks under the *noise-first* configuration.

7.2.1 Coffee Delivery. Figure 6 shows results for COFFEE, the simplest task. PROB-IRM consistently completes the task, matching the Perfect RM baseline performance (RQ1, RQ4). The primary difference is convergence time: PROB-IRM takes longer as it must first learn an accurate RM (RQ4). Higher noise levels further increase convergence time, as RM learning becomes more challenging (RQ5).

Both RECURRENT PPO and ISA fail to match this performance. The former never achieves the goal consistently (RQ2), while the latter performs well in the noiseless case but degrades substantially under noise (RQ3).

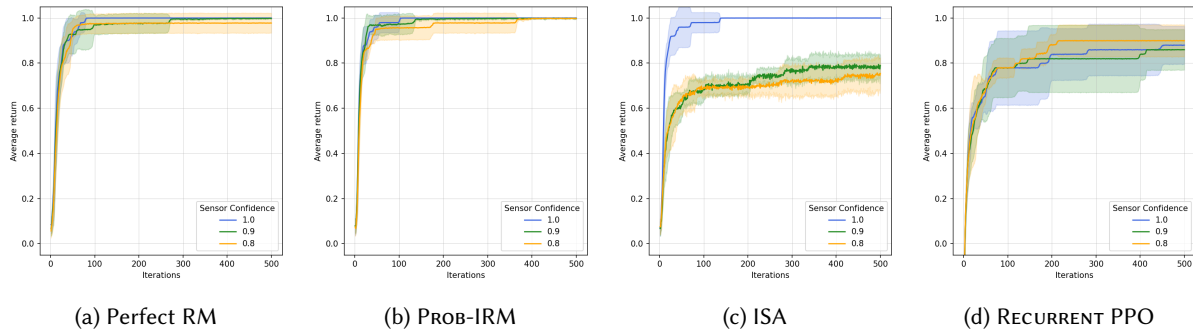


Fig. 6. Learning curves for COFFEE across different sensor confidence levels.

Table 1. Hyperparameter values. Search ranges show values tested during random search; N/A indicates fixed values.

Hyperparameter	Value	Search Range
Environment		
Max. episode length		
OFFICEWORLD (COFFEE, COFFEEMAIL)	500	N/A
OFFICEWORLD (VISITABCD)	1,000	N/A
WATERWORLD (all)	150	N/A
PPO		
Total batch size	32,768	N/A
Learning rate	0.000153	log-uniform $[10^{-8}, 10^{-3}]$
Minibatch size	8,192	{1,024, 2,048, 4,096, 8,192}
GAE lambda	0.95	uniform [0.9, 1]
PPO clip range	0.2	{0.1, 0.2, 0.3}
Num. epochs	13	{5, ..., 15}
Value loss coeff.	0.5	uniform [0.5, 1]
Entropy loss coeff.	0.1	{0.1, 1}
Discount (γ)	0.99	N/A
Value clip	10.0	N/A
Gradient clip	0.5	log-uniform $[10^{-1}, 10^1]$
Gradient clipping	Global norm	N/A
Network		
Num. hidden layers	4	{1, 2, 4}
Units per hidden layer	64	{8, 16, 32, 64, 128}
Activation	ReLU	{ReLU, Tanh, ELU}
Share actor-critic layers	True	{True, False}
Weight initialiser	Orthogonal	N/A
LSTM size (RECURRENT PPO)	32	{8, 16, 32}
PROB-IRM		
Max. number of RM states ($ U_{\max} $)	10	N/A
Max. edges per state pair (κ)	1	N/A
Optimiser	Adam	N/A
Cross-entropy threshold (β)	0.6	{0, 0.5, 0.6, 0.8, 1, 1.5, 2}
Example penalty budget	3,200	log-uniform $[10^2, 10^4]$
Min. example penalty threshold (t_{epen})	2	N/A
Num. grounded traces per noisy trace ($n_{\lambda'}$)	10	N/A
Trace origin mult. (w_{t_o})	0.25	{0.25, 0.5, 1, 1.5}
Reward shaping	True	{True, False}

7.2.2 Coffee and Mail Delivery. Figure 7 shows results for COFFEEMAIL. PROB-IRM again achieves performance comparable to the Perfect RM baseline across all noise levels (RQ1, RQ4).

As in COFFEE, RECURRENT PPO fails to achieve the task goal (RQ2), while ISA follows the same pattern, performing well in the noiseless case but obtaining even lower performance under noise (RQ3).

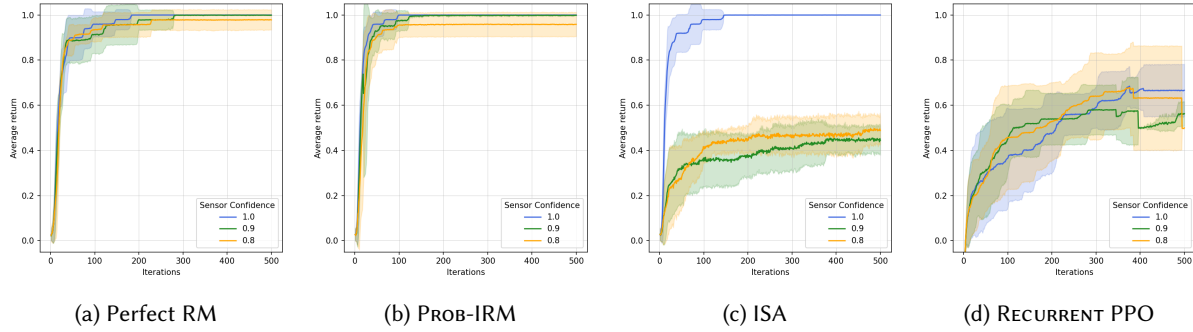


Fig. 7. Learning curves for COFFEEMAIL across different sensor confidence levels.

7.2.3 Sequential Visitation. Figure 8 shows results for VISITABCD, the most challenging task. PROB-IRM consistently achieves the task’s goal for sensor confidences of 1.0 and 0.9, but degrades at 0.8 due to one particularly difficult seed (RQ1, RQ4, RQ5).

RECURRENT PPO again fails to solve the task at any noise level (RQ2). More notably, ISA fails to converge under noise despite performing well in the noiseless case (RQ3). This degradation is more pronounced than in simpler tasks, highlighting the growing importance of noise-aware RM learning as task complexity increases.

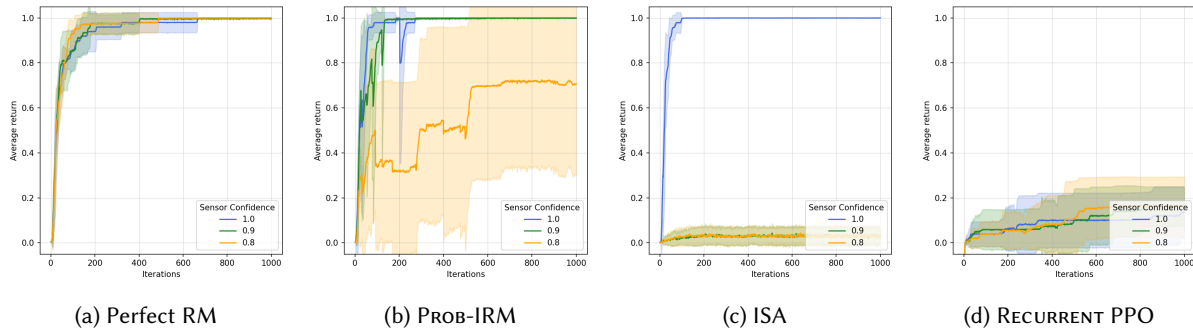


Fig. 8. Learning curves for VISITABCD across different sensor confidence levels.

7.2.4 Summary. PROB-IRM consistently solves all three OFFICEWORLD tasks (RQ1), substantially outperforming RECURRENT PPO and ISA (RQ2, RQ3). The former fails to solve the tasks across all noise levels, while the latter, despite performing well in noiseless settings, degrades under noise as task complexity grows.

PROB-IRM is comparable to Perfect RM at lower noise levels (0.9, 1.0), with performance degrading at higher noise (0.8) on harder tasks (RQ4, RQ5). The primary trade-off is increased convergence time, as the agent must first learn an accurate RM.

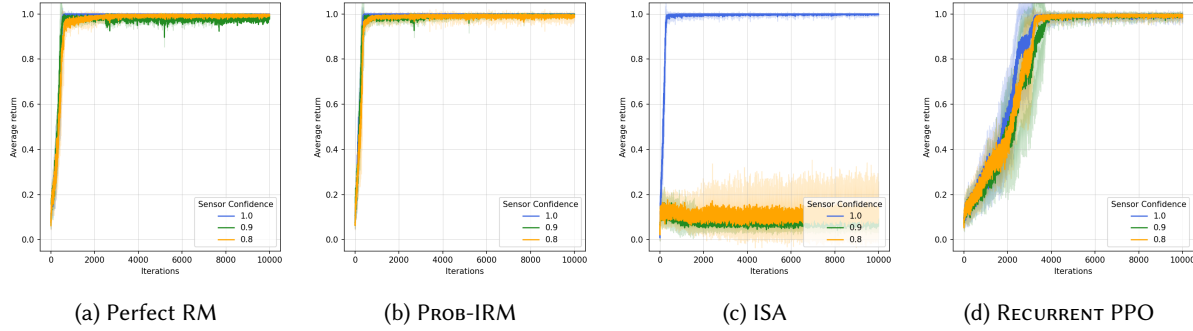


Fig. 9. Learning curves for RGB across different sensor confidence levels.

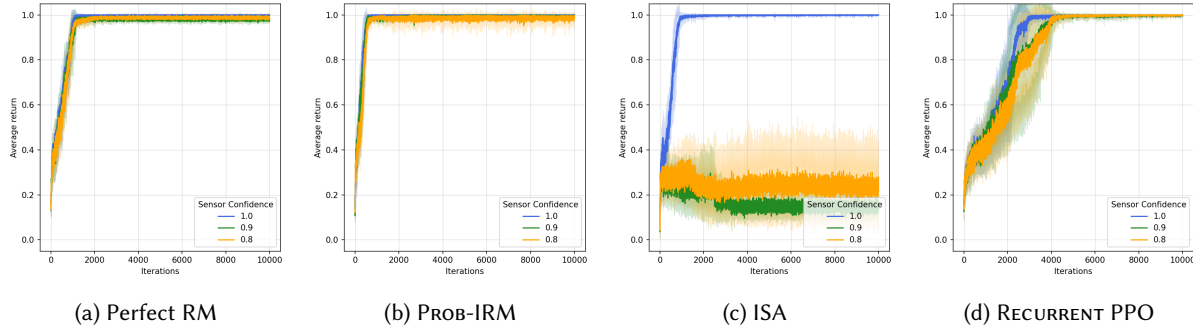


Fig. 10. Learning curves for RG-B across different sensor confidence levels.

Unlike our previous work (Parac et al. 2024), we observe stable RM relearning frequencies across noise levels. We attribute this to collecting more traces between relearning checks, yielding more accurate estimates of the underlying distribution and consequently more stable RM learning.

7.3 WATERWORLD Main Results

We evaluate PROB-IRM and baselines on the three tasks of the WATERWORLD domain under the *noise-first* configuration, addressing RQ6 on scalability to continuous state spaces.

7.3.1 RGB. Figure 9 shows results for RGB. PROB-IRM solves the task across all noise levels, converging close to the Perfect RM baseline (RQ1, RQ4, RQ5). RECURRENT PPO converges significantly more slowly (RQ2), while ISA follows the pattern observed in OFFICEWORLD: good performance in the noiseless case but substantial degradation under noise (RQ3).

7.3.2 RG-B. Figure 10 shows results for RG-B. PROB-IRM matches the Perfect RM baseline in both final return and convergence speed (RQ4). RECURRENT PPO converges more slowly but eventually solves the task (RQ2), while ISA again degrades under noise (RQ3).

7.3.3 RGBC. Figure 11 shows results for RGBC, the most challenging task. PROB-IRM solves it across all noise levels (RQ1, RQ4), while RECURRENT PPO fails to make progress regardless of noise level (RQ2).

Notably, ISA fails even in the noiseless setting (RQ3), contrasting with results in Furelos-Blanco, Law, Jonsson, et al. (2021). We attribute this to our distributed training architecture: rather than checking for RM relearning after each trace, we check at iteration end, adding all conflicting examples simultaneously. The resulting large learning task causes ISA to time out.

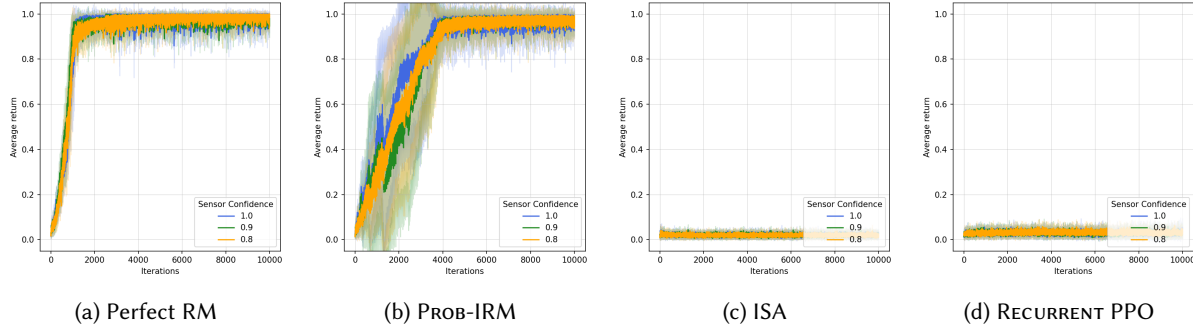


Fig. 11. Learning curves for RGBC across different sensor confidence levels.

7.4 Ablations

We conduct ablation studies to assess the impact of key design choices in PROB-IRM.

7.4.1 Probabilistic Reward Shaping. Figure 12 compares PROB-IRM with and without probabilistic reward shaping (see Section 5.3) on COFFEE. Both variants reach similar final performance, but reward shaping accelerates convergence and achieves slightly higher returns. This speed-up occurs because shaped rewards provide intermediate signals upon partial task completion.

Unlike our previous work (Parac et al. 2024), we observe little difference in this benefit across noise levels, and in all cases, a single RM learning iteration suffices to find a robust RM. We attribute this to collecting more traces before learning the first RM, reducing reliance on reward shaping to generate informative counterexamples.

7.4.2 Multiple Noise Sources. We assess PROB-IRM’s robustness to multiple independent noise sources using the *noise-all* configuration on COFFEE, where $\clubsuit$, $*$, and o sensors are all noisy, tripling noise exposure compared to *noise-first* (RQ4).

Figure 13 shows the results. Surprisingly, PROB-IRM outperforms the Perfect RM baseline. This occurs because the learned RM is more robust to noise accumulation: with both $\clubsuit$ and o sensors noisy, there is a non-zero probability at each step of erroneously transitioning to u_A from u_0 . As this probability accumulates over the episode, it may cause premature termination. The learned RM mitigates this by slowing probability mass accumulation towards u_A , reducing spurious terminations and simplifying policy learning.

7.4.3 Pre-trained Noise Model. To evaluate PROB-IRM under a different noise model, we replace the probabilistic sensors with pre-trained linear classifiers on COFFEE. We collect observation-label pairs from 1,000 random episodes as training data, with the resulting sensor F1 scores in Table 2.

This noise model differs substantially from our main experiments: sensor confidence varies across events, noise includes both false positives and false negatives, and any event may be detected regardless of task relevance.

Figure 14 compares learning curves for Perfect RM and PROB-IRM. Both converge to suboptimal performance, confirming this noise model is challenging. Despite this, PROB-IRM is only slightly worse than the Perfect RM baseline, demonstrating effectiveness under more complex noise models.

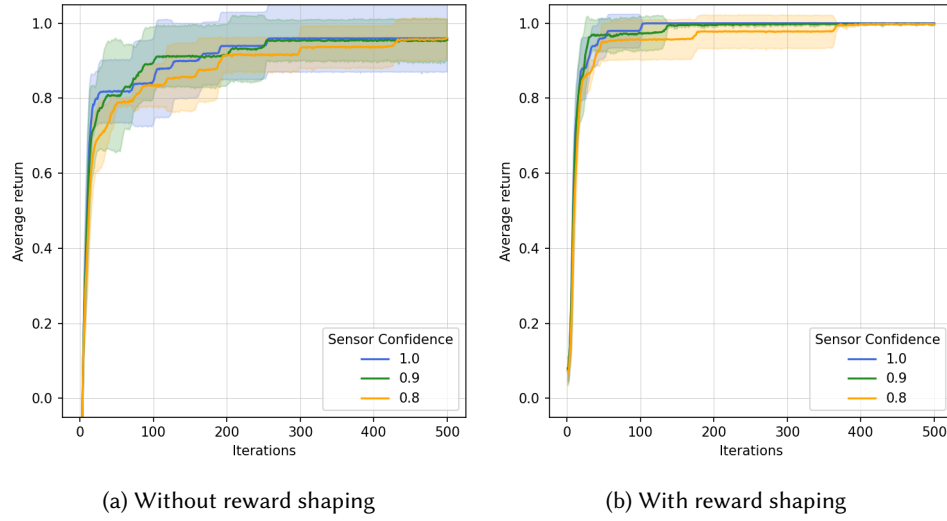


Fig. 12. PROB-IRM learning curves for COFFEE with and without probabilistic reward shaping.

Table 2. F1 scores for each sensor in the pre-trained sensor model ablation (10 runs).

Sensor		A	B	C	D	☕	✉	o	*
F1 Score	Min.	0.60	0.70	0.80	0.70	0.87	0.80	0.60	0.40
	Max.	1.00	1.00	1.00	1.00	1.00	1.00	0.90	0.80
	Avg.	0.84	0.90	0.89	0.92	0.94	0.95	0.72	0.59
	Std. Dev.	0.14	0.10	0.07	0.10	0.05	0.07	0.10	0.14

7.4.4 Task Length. To evaluate PROB-IRM’s scalability to longer tasks, we generalise VISITABCD to require visiting increasingly long sequences of locations. All agents use sensors at 80% confidence (our lowest tested level).

Figure 15 shows learning curves. Agents converge to high performance for sequences up to length 3. Performance begins degrading at length 4 and drops drastically at length 5. This is expected: the probability of observing an error-free goal trace decreases exponentially with task length. Without sufficient clean goal traces, learning a correct RM becomes increasingly difficult.

7.4.5 Belief Updating vs Thresholding. We provide empirical justification for our belief updating approach (Section 5.1) over thresholding. Thresholding treats any proposition with probability above a fixed threshold as true, avoiding belief maintenance but introducing a critical drawback: under our sensor model, it either works perfectly or fails to learn any RM.

We train PROB-IRM agents on COFFEE under *noise-all* with 80% sensor confidence, using thresholds of 0.7 (below) and 0.9 (above). Figure 16 shows the results. The 0.7 threshold achieves good performance by correctly identifying propositions. However, the 0.9 threshold fails: it incorrectly rejects *every* proposition, making RM learning impossible.

This highlights thresholding’s brittleness: success requires either precise knowledge of the noise posterior or careful threshold tuning, which are both restrictive in practice. In contrast, belief updating requires no such

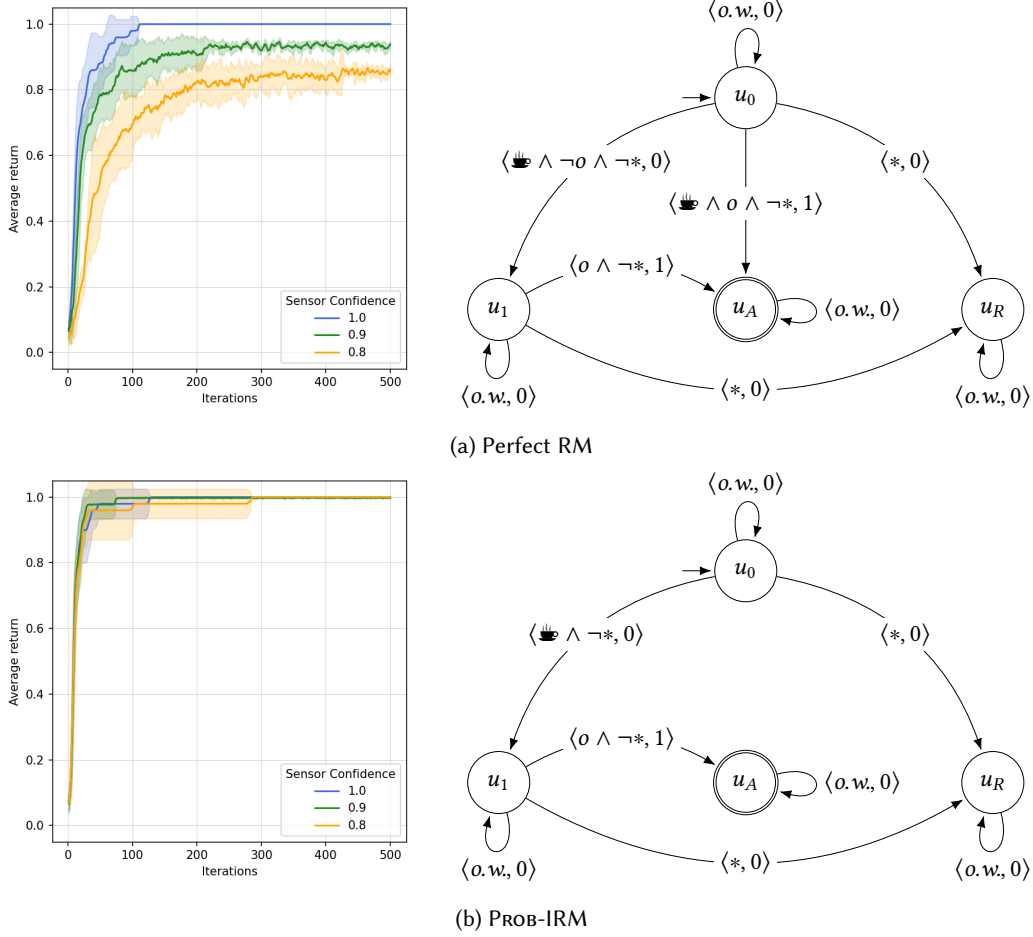


Fig. 13. Learning curves and exploited RMs for COFFEE in the *noise-all* setting.

knowledge, introduces no additional hyperparameters, and achieves strong performance in the same setting (Figure 6).

7.4.6 Cross-Entropy Threshold. We evaluate the impact of the cross-entropy relearning threshold β on COFFEMAIL, where RM relearning is triggered multiple times. We test values from 0 (relearn every iteration) to 2 (relearn when the model assigns less than 13.5% probability to the correct class).

Figure 17 shows the results. Performance is generally insensitive to this parameter, benefiting from ILASP’s scoring function as a verification step after RM learning (Section 6.1). This verification prevents updates that do not improve example coverage, making the threshold less critical.

7.4.7 Balancing Example Coverage and RM Simplicity. The ILASP scoring function (Definition 2.3) balances hypothesis length against uncovered example penalties. By adjusting the total penalty budget, we control the trade-off between example coverage and RM simplicity.

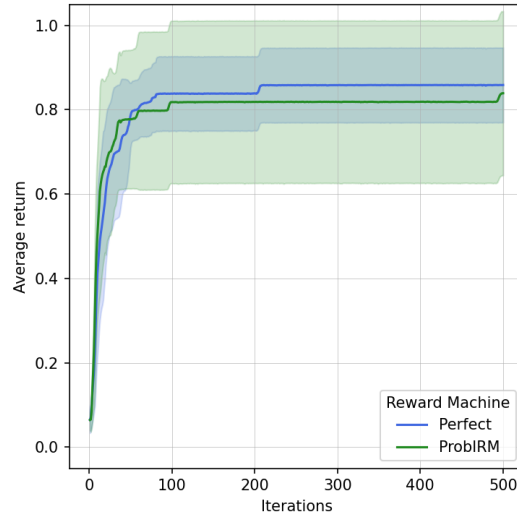


Fig. 14. Learning curves for COFFEE with a pre-trained noise model.

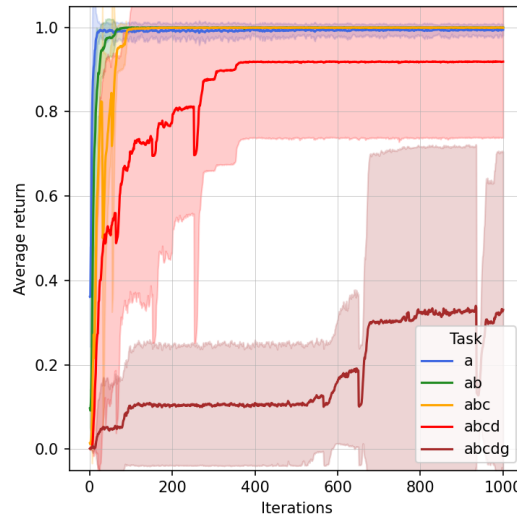


Fig. 15. Learning curves for VISITABCD tasks with varying sequence lengths (80% sensor confidence).

To evaluate this trade-off, we adjust the length function and the example penalty budget (Section 4.3.4). For the former, the lengths of ed and $\hat{\phi}$ rules are set to 2 and 1, respectively. For the latter, low budgets cause low-penalty examples to be ignored since their penalty rounds to 0. To ensure a fair comparison, we scale the WCDPI example inclusion threshold t_{epen} proportionally to the budget, preventing higher budgets from disproportionately including more examples.

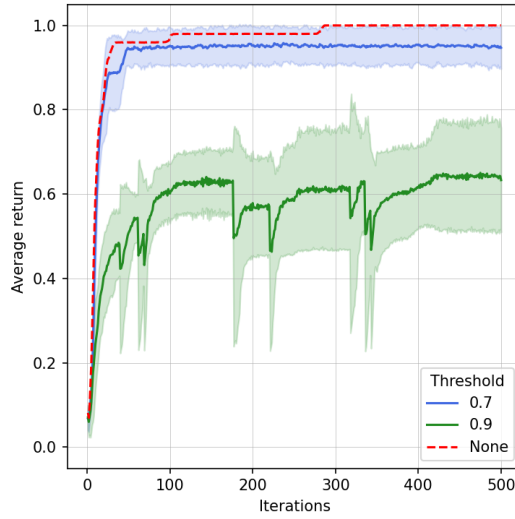


Fig. 16. Learning curves for COFFEE (*noise-all*, 80% confidence) comparing thresholding at 0.7 and 0.9 with belief updating (dashed line).

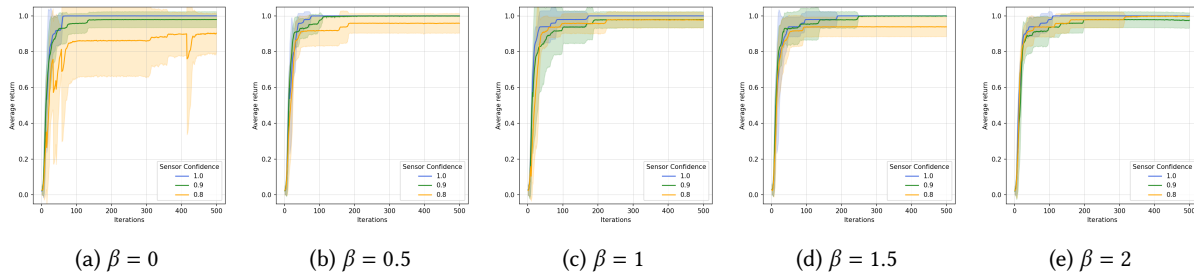


Fig. 17. PROB-IRM learning curves for different values of the cross-entropy threshold β .

Figure 18 shows results for COFFEE_{MAIL}. Low budgets (100, 200) over-emphasise compactness, yielding RMs that fail to solve the task reliably. Higher budgets achieve better performance but with increased variance. Our default is 3,200, the smallest value for which ILASP consistently finds solutions within the 60-minute timeout.

7.4.8 Class-Based Weight Rebalancing. We evaluate the impact of class-based weight rebalancing on COFFEE. We compare the rebalancing procedure from Section 4.3.3, which sets $w_c = \frac{1}{n_c}$ where n_c is the number of examples of class $c \in \{\text{goal, dead-end, incomplete}\}$, against a baseline where $w_c = 1$ for all examples.

Figure 19 shows that rebalancing improves both convergence speed and stability. Without rebalancing, the learned RM initially captures only negative transitions (avoiding *), since dead-end examples are more likely than goal ones under a random policy. As the agent learns to avoid *, the example distribution becomes more informative, but rebalancing accelerates this by ensuring goal examples are adequately weighted.

7.4.9 Example Generation from Previous RMs. We evaluate the impact of generating examples from previously learned RMs (Section 4.1.2). Figure 20 compares PROB-IRM with ($w_{t_o} = 0.25$) and without ($w_{t_o} = 0$) this technique

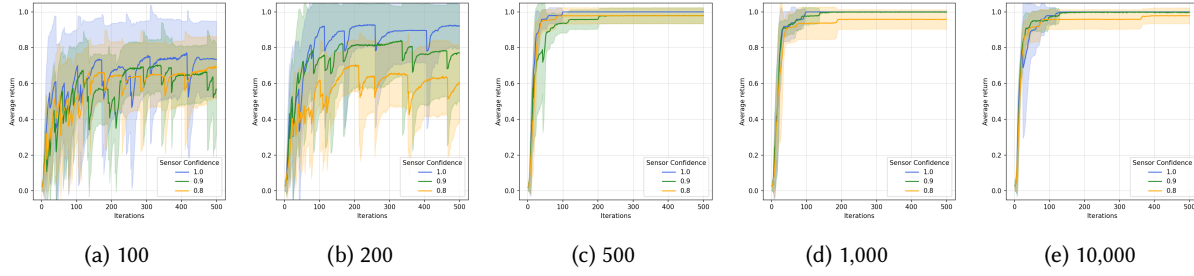
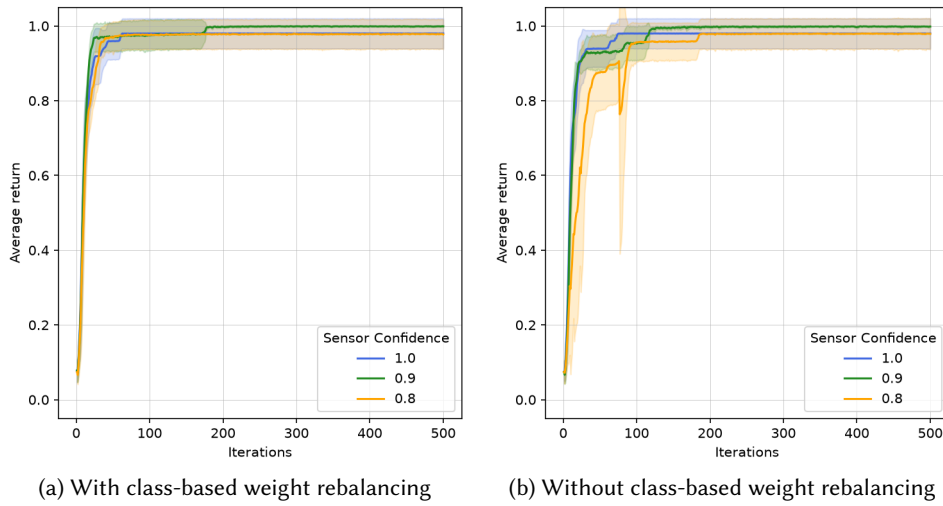
Fig. 18. PROB-IRM learning curves for different example *penalty budgets* in COFFEEMAIL.

Fig. 19. PROB-IRM learning curves for COFFEE when class-based weight rebalancing is enabled and disabled.

on RGBC under 80% sensor confidence. Including examples from previous RMs substantially improves convergence speed, confirming the effectiveness of this approach.

7.5 RM Learning Computational Overhead

We evaluate ILASP runtime as a percentage of total execution time across all PROB-IRM experiments. Table 3 summarises the results.

For COFFEE and COFFEEMAIL, ILASP cost is negligible across all noise levels, remaining below 1% and typically below 0.01%. For VISITABCD, cost is similarly low at 90% and 100% sensor confidence, but increases substantially at 80%, where ILASP accounts for roughly 25% of total runtime on average, reaching 75% in the most challenging instances. WATERWORLD tasks show the same trend.

In general, ILASP cost correlates positively with both task complexity and noise level. Two factors drive this: first, as training progresses, the growing number of examples makes each ILASP task harder to solve. Second, ILASP's optimality guarantee (Definition 2.3) requires progressively increasing a score threshold until a solution is found, meaning harder tasks require more iterations.

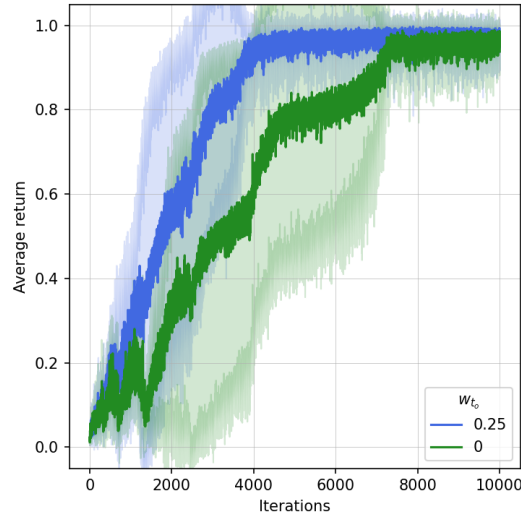


Fig. 20. PROB-IRM learning curves for RGBC under 80% sensor confidence with ($w_{t_0} = 0.25$) and without ($w_{t_0} = 0$) example generation from previous RMs.

8 Related Work

In this section, we discuss work related to PROB-IRM, organised into four areas: learning RMs without noise (Section 8.1), uncertainty-aware RM exploitation (Section 8.2), learning RMs under uncertainty (Section 8.3), and inductive logic programming with noisy examples (Section 8.4).

8.1 Learning Reward Machines without Noise

Since Toro Icarte, Klassen, R. A. Valenzano, et al. (2018) introduced reward machines, numerous methods have been proposed for learning them from traces, including active learning (Gaon and Brafman 2020), discrete optimisation (Christoffersen et al. 2020; Toro Icarte, Klassen, R. Valenzano, et al. 2023; Toro Icarte, Waldie, et al. 2019), inductive logic programming (Ardon, Furelos-Blanco, Parać, et al. 2025; Ardon, Furelos-Blanco, and Russo 2024; Furelos-Blanco, Law, Jonsson, et al. 2023, 2021; Furelos-Blanco, Law, Russo, et al. 2020), program synthesis (H. Hasanbeig et al. 2024; M. Hasanbeig, Jeppu, et al. 2021), SAT solving (Corazza et al. 2022; Xu et al. 2020), and state-merging (Gaon and Brafman 2020). However, these methods assume deterministic labelling functions and cannot handle label noise.

We build upon Furelos-Blanco, Law, Jonsson, et al. (2021), which uses ILASP (Law et al. 2015), an inductive logic programming system whose weighted example mechanism makes it well suited for extension to noisy settings. However, this approach learns minimal RMs through iterative deepening, a property we cannot maintain under noise as discussed in Section 6.2. Additionally, the method can suffer from overgeneralisation, which we describe in Section 9.

8.2 Uncertainty-Aware Exploitation of Reward Machines

Traditional RM exploitation methods assume perfect labelling functions. These include QRM (Toro Icarte, Klassen, R. A. Valenzano, et al. 2018), CRM (Toro Icarte, Klassen, R. A. Valenzano, et al. 2022), and hierarchical RL approaches (Furelos-Blanco, Law, Jonsson, et al. 2023, 2021; Toro Icarte, Klassen, R. A. Valenzano, et al. 2022, 2018).

Table 3. ILASP runtime as a percentage of total training time across tasks and sensor confidence levels.

Domain	Noise	Min.	Median	Max.	Average	Std. Dev.
COFFEE	80%	<0.01%	<0.01%	<0.01%	<0.01%	<0.01%
	90%	<0.01%	<0.01%	<0.01%	<0.01%	<0.01%
	100%	<0.01%	<0.01%	<0.01%	<0.01%	<0.01%
COFFEEMAIL	80%	0.05%	0.10%	0.96%	0.31%	0.31%
	90%	0.01%	0.05%	0.96%	0.21%	0.34%
	100%	<0.01%	0.01%	0.05%	0.02%	0.01%
VISITABCD	80%	0.04%	20.32%	75.42%	27.14%	21.69%
	90%	0.02%	0.08%	8.54%	1.75%	3.11%
	100%	0.05%	0.20%	0.26%	0.18%	0.07%
RGB	80%	<0.01%	2.36%	7.10%	2.40%	2.31%
	90%	<0.01%	0.02%	1.09%	0.23%	0.43%
	100%	0.02%	0.04%	0.08%	0.05%	0.02%
RG-B	80%	<0.01%	0.29%	2.23%	0.56%	0.83%
	90%	<0.01%	0.42%	1.31%	0.53%	0.52%
	100%	0.42%	1.28%	2.40%	1.38%	0.70%
RGBC	80%	0.07%	0.48%	30.66%	5.83%	11.27%
	90%	0.28%	0.41%	15.24%	2.99%	5.32%
	100%	0.20%	0.78%	13.53%	3.90%	4.85%

Recent work uses deep RL algorithms such as PPO, encoding the current RM state via one-hot vectors (Ardon, Furelos-Blanco, Parac, et al. 2025) or graph embeddings (Furelos-Blanco, Pert, et al. 2026; Yalcinkaya et al. 2023, 2024). However, these algorithms suffer significant performance degradation under sensor noise due to erroneous labels causing agents to skip necessary subtasks or pursue incorrect objectives (Nodari 2023).

Li, Z. Chen, Klassen, et al. (2024) and Li, Z. Chen, Vaezipoor, et al. (2022) evaluate several exploitation strategies under the noisy setting formalised in Definition 3.2, including thresholding (converting probabilistic labels to deterministic ones by treating any proposition with probability above 0.5 as occurring), independent belief updating (maintaining a distribution over RM states), recurrent policies (bypassing labels entirely), and RNN-based belief estimation (which requires access to perfect labels during training). Li, Z. Chen, Vaezipoor, et al. (2022) combine these with QRM, while Li, Z. Chen, Klassen, et al. (2024) use PPO. We adopt the independent belief updating approach, conditioning PPO on the current RM state belief vector (Section 5.1), and compare against thresholding (Section 7.4.5) and recurrent policies (Sections 7.2 and 7.3) in our experiments.

M. Hasanbeig, Kantaros, et al. (2019) also consider label uncertainty within a broader framework for uncertain transition dynamics. While their formulation permits probabilistic labelling, it does not explicitly include labelling or termination functions.

8.3 Learning Reward Machines under Uncertainty

Several approaches have addressed uncertainty in different components of the RM learning problem. Dohmen et al. (2022) address uncertainty in RM transitions, learning RMs with probabilistic transition functions by adapting the L^* active learning algorithm (Angluin 1987). Corazza et al. (2022) address uncertainty in rewards, learning

RMs with stochastic reward emissions. In contrast, we address uncertainty in labels, learning deterministic RMs from noisy labels.

The work most similar to ours is due to Verginis et al. (2024), who also tackle noisy labels. They adapt JIRP (Xu et al. 2020), a SAT-based approach that learns RMs by iteratively covering counterexamples, similarly to the work we build upon (Furelos-Blanco, Law, Jonsson, et al. 2021). Since JIRP does not handle uncertainty, they use thresholding to convert noisy labels to deterministic ones while simultaneously learning an improved labelling function estimate. To handle inconsistencies between candidate RMs and newly observed traces, they maintain two labelling function estimates: one for generating learning examples and one representing the best estimate so far. When these diverge significantly, they restart the learning process. Their approach thus focuses on learning an improved labelling function to approximate the true one, whereas we directly integrate noise into the RM learning procedure through weighted examples.

While their labelling function learning could potentially be combined with our approach, their Bayesian update mechanism requires tracking repeated state visits, which complicates application to continuous state spaces without discretisation.

8.4 Inductive Logic Programming with Noisy Examples

Several works (Cunnington, Cipcigan, et al. 2023; Cunnington, Law, et al. 2023, 2024; Dreossi et al. 2024) use ILASP to learn logical rules from noisy examples in domains beyond RM learning. Like our approach, Cunnington, Law, et al. (2023) aggregate multiple sources of noise for a single example. However, they use t-norm operators (Klement et al. 2000) to combine probabilistic predictions, whereas we use a sampling-based approach with uniform penalty weights.

9 Discussion

In this section, we discuss the limitations of PROB-IRM.

9.1 Belief Overestimation from Temporally Correlated Noisy Labels

Our RM state belief update (Section 5.1) treats each noisy label as providing new independent evidence for RM state transitions. However, as noted by Li, Z. Chen, Klassen, et al. (2024), when agents revisit the same state, this causes belief overestimation: noisy labels from the same state reflect the same underlying reality, yet the update rule treats them as independent samples, causing belief to spuriously accumulate.

EXAMPLE 14. Consider extending the history and trace from Example 11 with three additional visits to the same state:

$$h = \langle \langle 4, 6 \rangle, \text{left}, \langle 3, 6 \rangle, \text{left}, \langle 3, 6 \rangle, \text{left}, \langle 3, 6 \rangle \rangle,$$

$$\tilde{\lambda} = \langle \{ \text{⚡} : 0.0002, o : 0.0001 \}, \{ \text{⚡} : 0.6513, o : 0.0001 \}, \{ \text{⚡} : 0.6513, o : 0.0001 \}, \{ \text{⚡} : 0.6513, o : 0.0001 \} \rangle.$$

Computing the RM state belief evolution using Definition 5.1, we have:

- Step $t = 0$. The agent starts in environment state $s_0 = \langle 4, 6 \rangle$. We compute the initial belief using the noisy label $\tilde{\mathcal{L}}(\emptyset, \emptyset, s_0) = \{ \text{⚡} : 0.0002, o : 0.0001 \}$. From Example 11:

$$\tilde{u}_0 \approx \begin{bmatrix} 0.9998 & 0.0002 & 0.0000 & 0 \end{bmatrix}^\top.$$

- Step $t = 1$. The label is $\tilde{\mathcal{L}}(s_0, a_0, s_1) = \{ \text{⚡} : 0.6513, o : 0.0001 \}$, where $s_0 = \langle 4, 6 \rangle$, $a_0 = \text{left}$, $s_1 = \langle 3, 6 \rangle$. From Example 11:

$$\tilde{u}_1 \approx \begin{bmatrix} 0.3486 & 0.6513 & 0.0001 & 0 \end{bmatrix}^\top.$$

- Step $t = 2$. The agent remains at $\langle 3, 6 \rangle$ and observes the same noisy label. As such, the resulting belief is:

$$\tilde{u}_2 \approx [0.1215 \quad 0.8783 \quad 0.0002 \quad 0]^\top.$$

- Step $t = 3$. The agent remains at $\langle 3, 6 \rangle$ again. The same noisy label yields:

$$\tilde{u}_3 \approx [0.0424 \quad 0.9573 \quad 0.0003 \quad 0]^\top.$$

Despite receiving no new information (the agent remained in the same state), belief in u_1 increased from 0.6513 to 0.9573 due to treating temporally correlated labels as independent.

Li, Z. Chen, Klassen, et al. (2024) address this by training a recurrent neural network to model temporal dependencies in noisy labels. However, their approach requires access to perfect labels during training, limiting applicability to fully noisy settings. Despite not explicitly addressing this limitation, our experiments show minimal practical impact.

9.2 Overgeneralisation from Temporally Dependent Propositions

A second limitation arises when propositions exhibit temporal dependencies. In these cases, the learned RM may be *valid* with respect to all observed traces (Section 2.2) yet fail to capture causally necessary propositions, causing *overgeneralisation*.

EXAMPLE 15. Consider a modified COFFEE (Example 1) where the office (o) can only be reached after having been in the coffee location (☕) first. PROB-IRM, like methods learning minimal RMs (e.g., Furelos-Blanco, Law, Jonsson, et al. 2021; Xu et al. 2020), would learn an RM containing only the office transition, omitting coffee entirely despite it being necessary. The learned RM accepts all observed traces but overgeneralises to traces the environment cannot produce.

Interestingly, noisy detections may inadvertently help by breaking strict temporal dependencies between propositions. However, a more principled solution requires generating negative examples: traces that would be accepted by the candidate RM but cannot occur in the environment. H. Hasanbeig et al. (2024) generate such examples up to a tunable length l , repeating learning until no negative examples remain. A similar strategy could extend our approach, though selecting an appropriate l is task-dependent and non-trivial. A more general solution may require explicit causal modelling of proposition dependencies, which we leave for future work.

9.3 Convergence and Cyclic Relearning

Our approach builds upon ISA (Furelos-Blanco, Law, Jonsson, et al. 2021), which provably converges to a target RM in a finite number of steps (Theorem 7.1). However, a key difference distinguishes our method: while ISA uses unweighted examples that must all be satisfied, PROB-IRM uses weighted examples with finite penalties. This allows the learner to trade off example coverage against RM complexity, but introduces a potential failure mode: previously covered examples can become uncovered by subsequent RMs. We demonstrate that this can cause infinite cyclic relearning, even in the noiseless setting.

Figure 21 illustrates a simplified OFFICEWORLD task where the agent must visit either A or B , moving only left or right. We set the ILASP example penalty sum (Section 7.4.7) to 10 and the cost of a rule representing an edge to 6. For simplicity, we assume the agent alternates between the actions *left* and *right* across episodes. We show that PROB-IRM cycles indefinitely between $\mathcal{RM}_1$ (Figure 21c) and $\mathcal{RM}_2$ (Figure 21d):

- (1) Initially, the agent goes left, reaches A , and terminates. The following WCDPI is formed:

$$\left\langle \text{ex1}, 10, \langle \{\text{accept}\}, \{\text{reject}\} \rangle, \left\{ \begin{array}{l} \text{prop}(A, \emptyset). \\ \text{step}(\emptyset). \text{last}(\emptyset). \end{array} \right\} \right\rangle.$$

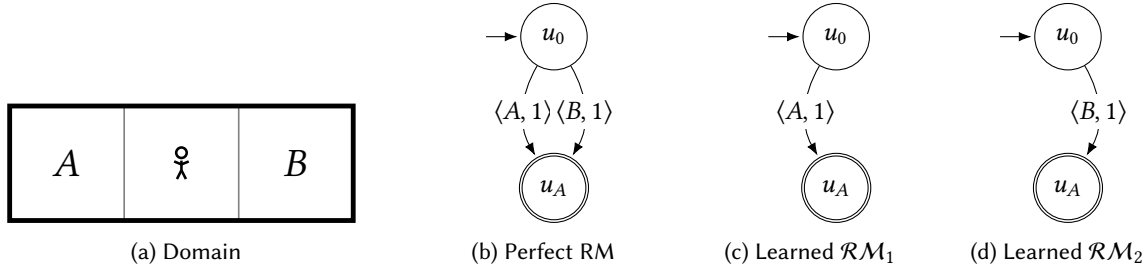


Fig. 21. Simplified OFFICEWORLD task (Visit A or B) demonstrating cyclic relearning. The perfect RM accepts both propositions, but PROB-IRM alternates between $\mathcal{RM}_1$ (accepts only A) and $\mathcal{RM}_2$ (accepts only B) when the edge cost exceeds the example penalty.

The example has penalty 10. Recall that penalties are proportional to the relative frequency of the example. ILASP learns $\mathcal{RM}_1$ from this example.

- (2) The agent goes right, reaches B, and terminates. The learning task now contains two examples, each with penalty 5:

$$\left\langle \text{ex1}, 5, \langle \{\text{accept}\}, \{\text{reject}\} \rangle, \left\{ \begin{array}{l} \text{prop}(A, \emptyset). \\ \text{step}(\emptyset). \text{last}(\emptyset). \end{array} \right\} \right\rangle,$$

and

$$\left\langle \text{ex2}, 5, \langle \{\text{accept}\}, \{\text{reject}\} \rangle, \left\{ \begin{array}{l} \text{prop}(B, \emptyset). \\ \text{step}(\emptyset). \text{last}(\emptyset). \end{array} \right\} \right\rangle.$$

The perfect RM (accepting both) would require adding an edge (cost 6), while ignoring ex2 costs only 5. Since $\mathcal{RM}_1$ already satisfies this constraint, it is retained. Note that $\mathcal{RM}_2$ would be equally optimal, but RMs are only updated when the new candidate achieves a strictly better score.

- (3) The agent goes left, reaches A, and terminates. The current RM ($\mathcal{RM}_1$) captures the trace, so no relearning occurs.
- (4) The agent goes right, reaches B, and terminates. Another trace for ex2 is produced and its penalty (7) grows large enough that ignoring it becomes costlier than adding an edge. ILASP then learns $\mathcal{RM}_2$.
- (5) Subsequently, the agent alternates between producing ex1 and ex2, with class rebalancing causing penalties to oscillate and triggering cyclic relearning as shown in Table 4.

This cycling behaviour is a direct consequence of using finite penalty budgets. With an infinite penalty budget, all examples would carry infinite weight, requiring complete coverage and reducing PROB-IRM to ISA, which provably converges (Furelos-Blanco, Law, Jonsson, et al. 2021). The finite penalty budget enables trading off RM complexity against example coverage, but introduces the possibility of cyclic relearning.

Although infinite cycling is theoretically possible, it is unlikely in practice for several reasons. Even under a policy and hyperparameter configuration specifically designed to produce traces contradicting the current RM, an exponentially increasing number of traces is required to induce a meaningful shift in WCDPI penalties. Moreover, trained RL policies tend to exploit narrow trajectory distributions in later training stages, further reducing trace diversity and stabilising example distributions.

Table 4. Example penalties and examples required for RM learning iterations in the cycling scenario. Column “Additional traces needed to trigger relearning” captures the number of new traces collected between iterations $i - 1$ and i ; Y is a variable representing the number of ex2-generating traces at iteration $2n - 1$.

#	RM learned	Penalty of ex1	Penalty of ex2	Number of traces generating ex1	Number of traces generating ex2	Additional traces needed to trigger relearning
1	$\mathcal{RM}_1$	$\frac{1}{1} \times 10 = 10$	—	1	0	1
2	$\mathcal{RM}_2$	$\frac{1}{1} \times 10 \approx 3$	$\frac{1}{2} \times 10 \approx 7$	1	2	2
3	$\mathcal{RM}_1$	$\frac{1}{2} \times 10 = 6$	$\frac{1}{2} \times 10 = 4$	3	2	2
4	$\mathcal{RM}_2$	$\frac{1}{2} \times 10 \approx 4$	$\frac{1}{2} \times 10 \approx 6$	3	4	2
5	$\mathcal{RM}_1$	$\frac{1}{2} \times 10 \approx 6$	$\frac{1}{2} \times 10 \approx 4$	5	4	2
6	$\mathcal{RM}_2$	$\frac{5}{12} \times 10 \approx 4$	$\frac{7}{12} \times 10 \approx 6$	5	7	3
				$\vdots$		
10	$\mathcal{RM}_2$	$\frac{15}{34} \times 10 \approx 4$	$\frac{19}{34} \times 10 \approx 6$	15	19	7
				$\vdots$		
100	$\mathcal{RM}_2$	4	6	1.24×10^9	1.52×10^9	5.02×10^8
				$\vdots$		
1000	$\mathcal{RM}_2$	4	6	1.38×10^{87}	4.14×10^{87}	1.37×10^{87}
				$\vdots$		
$2n - 1$	$\mathcal{RM}_1$	6	4	$\lceil \frac{11}{9} \cdot Y \rceil$	Y	N/A^a
$2n$	$\mathcal{RM}_2$	4	6	$\lceil \frac{11}{9} \cdot Y \rceil$	$\lceil \frac{11}{9} \cdot \lceil \frac{11}{9} \cdot Y \rceil \rceil$	$\lceil \frac{11}{9} \cdot \lceil \frac{11}{9} \cdot Y \rceil \rceil - Y$ $\approx \lceil \frac{40}{81} Y \rceil$

^a Depends on the value at the previous row.

10 Conclusions and Future Work

We present PROB-IRM, a method for learning and exploiting reward machines from noisy labels. Our core contributions are: (1) a mechanism for learning RMs from noisy traces using weighted examples; (2) an algorithm interleaving RM and policy learning; and (3) a comprehensive evaluation on discrete and continuous domains.

PROB-IRM outperforms recurrent baselines and achieves performance comparable to handcrafted RMs. Key challenges addressed include: specifying the learning task (balancing RM complexity against trace coverage via weighted examples), generating training examples (sampling from noisy traces and previous RMs), and determining when to trigger relearning (via RM error thresholding). Compared to our previous work (Parac et al. 2024), we introduce the complete ILASP encoding, integrate with PPO, add example generation from previous RMs, extend to continuous domains, and significantly expand the evaluation.

This work opens several promising research directions:

- *Hierarchical and First-Order RMs.* Extending to RM hierarchies (Furelos-Blanco, Law, Jonsson, et al. 2023) and first-order RMs (Ardon, Furelos-Blanco, Parać, et al. 2025) could improve scalability and expressiveness.
- *Human-AI Collaboration.* Learned RMs could facilitate understanding human strategies and interpreting AI intentions. Collecting sufficient human interaction traces for RM learning is challenging, making methods that can learn from limited human data particularly valuable for enabling practical human-AI collaboration systems.

- *Instruction Following*. Recent work trains policies to generalise across instructions given as finite-state machines (Furelos-Blanco, Pert, et al. 2026; Yalcinkaya et al. 2023, 2024) or linear temporal logic (Jackermeier and Abate 2025; León et al. 2022; Qiu et al. 2023; Vaezipoor et al. 2021). Our mechanisms (RM state beliefs, probabilistic reward shaping, and noise-aware learning) could enable instruction following under noisy labels, allowing deployment on complex benchmarks such as CALVIN (Mees et al. 2022) with realistic sensor noise.
- *Labelling Function Learning*. Our work assumes the labelling function is given or pre-trained with a known proposition alphabet. Recent work learns the labelling function itself through environment interaction (Umili et al. 2024). Interleaving RM learning with labelling function learning is a key direction for future work.

Acknowledgments

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-222-0243. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorised to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Roko Parac is supported by the UKRI Centre for Doctoral Training in Safe and Trusted AI (EPSRC Project EP/S023356/1). We thank Julian de Gortari Briseno for his help with the noisy sensor model and Andrew Li for helpful advice about his work.

References

- D. Angluin. 1987. “Learning regular sets from queries and counterexamples.” *Information and Computation*, 75, 2, 87–106.
- L. Ardon, D. Furelos-Blanco, R. Parać, and A. Russo. 2025. “FORM: Learning Expressive and Transferable First-Order Logic Reward Machines.” In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.
- L. Ardon, D. Furelos-Blanco, and A. Russo. 2024. “Learning Reward Machines in Cooperative Multi-Agent Tasks.” In: *Autonomous Agents and Multiagent Systems. Best and Visionary Papers: AAMAS 2023 Workshops, Revised Selected Papers*, 43–59.
- A. Camacho, O. Chen, S. Sanner, and S. A. McIlraith. 2017. “Non-Markovian Rewards Expressed in LTL: Guiding Search Via Reward Shaping.” In: *International Symposium on Combinatorial Search (SOCS)*.
- A. Camacho, R. Toro Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. 2019. “LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning.” In: *International Joint Conference on Artificial Intelligence (IJCAI)*.
- P. J. K. Christoffersen, A. C. Li, R. Toro Icarte, and S. A. McIlraith. 2020. “Learning Symbolic Representations for Reinforcement Learning of Non-Markovian Behavior.” In: *Knowledge Representation and Reasoning Meets Machine Learning Workshop (KR2ML) at the Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- J. Corazza, I. Gavran, and D. Neider. 2022. “Reinforcement Learning with Stochastic Reward Machines.” In: *AAAI Conference on Artificial Intelligence (AAAI)*.
- D. Cunningham, F. Cipcigan, R. N. B. Ferreira, and J. Booth. 2023. “Symbolic Learning for Material Discovery.” In: *AI for Accelerated Materials Discovery Workshop (AI4Mat) at the Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- D. Cunningham, M. Law, J. Lobo, and A. Russo. 2023. “FFNSL: Feed-Forward Neural-Symbolic Learner.” *Machine Learning*, 112, 2, 515–569.
- D. Cunningham, M. Law, J. Lobo, and A. Russo. 2024. “The Role of Foundation Models in Neuro-Symbolic Learning and Reasoning.” In: *International Conference on Neurosymbolic Learning and Reasoning (NeSy)*.
- X. Ding, S. L. Smith, C. Belta, and D. Rus. 2011. “LTL Control in Uncertain Environments with Probabilistic Satisfaction Guarantees.” *IFAC Proceedings Volumes*, 44, 1, 3515–3520.
- T. Dohmen, N. Topper, G. K. Atia, A. Beckus, A. Trivedi, and A. Velasquez. 2022. “Inferring Probabilistic Reward Machines from Non-Markovian Reward Signals for Reinforcement Learning.” In: *International Conference on Automated Planning and Scheduling (ICAPS)*.
- T. Dreossi, A. Dovier, A. Formisano, M. Law, A. Manzato, A. Russo, and M. Tait. 2024. “Towards Explainable Weather Forecasting through FastLAS.” In: *International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*.
- G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. 2021. “Challenges of real-world reinforcement learning: definitions, benchmarks and analysis.” *Machine Learning*, 110, 9, 2419–2468.

- A. Eck, L. Soh, S. Devlin, and D. Kudenko. 2013. “Potential-Based Reward Shaping for POMDPs.” In: *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.
- J. Fu and U. Topcu. 2014. “Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints.” In: *Robotics: Science and Systems Conference (RSS)*.
- D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. 2023. “Hierarchies of Reward Machines.” In: *International Conference on Machine Learning (ICML)*.
- D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. 2021. “Induction and Exploitation of Subgoal Automata for Reinforcement Learning.” *Journal of Artificial Intelligence Research*, 70, 1031–1116.
- D. Furelos-Blanco, M. Law, A. Russo, K. Broda, and A. Jonsson. 2020. “Induction of Subgoal Automata for Reinforcement Learning.” In: *AAAI Conference on Artificial Intelligence (AAAI)*.
- D. Furelos-Blanco, C. Pert, F. Kelbel, A. F. Spies, A. Russo, and M. Dennis. 2026. “Beyond Fixed Tasks: Unsupervised Environment Design for Task-Level Pairs.” In: *AAAI Conference on Artificial Intelligence (AAAI)*.
- M. Gaon and R. I. Brafman. 2020. “Reinforcement Learning with Non-Markovian Rewards.” In: *AAAI Conference on Artificial Intelligence (AAAI)*.
- M. Ghasemi, E. Bulgur, and U. Topcu. 2020. “Task-Oriented Active Perception and Planning in Environments with Partially Known Semantics.” In: *International Conference on Machine Learning (ICML)*.
- H. Hasanbeig, N. Y. Jeppu, A. Abate, T. Melham, and D. Kroening. 2024. “Symbolic task inference in deep reinforcement learning.” *Journal of Artificial Intelligence Research*, 80, 1099–1137.
- M. Hasanbeig, N. Y. Jeppu, A. Abate, T. Melham, and D. Kroening. 2021. “DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning.” In: *AAAI Conference on Artificial Intelligence (AAAI)*.
- M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee. 2019. “Reinforcement Learning for Temporal Logic Control Synthesis with Probabilistic Satisfaction Guarantees.” In: *Conference on Decision and Control (CDC)*.
- C. Huang, Y. Li, C. C. Loy, and X. Tang. 2016. “Learning Deep Representation for Imbalanced Classification.” In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- M. Jackermeier and A. Abate. 2025. “DeepLTL: Learning to Efficiently Satisfy Complex LTL Specifications for Multi-Task RL.” In: *International Conference on Learning Representations (ICLR)*.
- A. Karpathy. 2015. *REINFORCEjs: WaterWorld demo*. (2015). <http://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html>.
- E. Klement, R. Mesiar, and E. Pap. 2000. *Triangular Norms*. Trends in Logic. Vol. 8. Springer.
- M. Law, A. Russo, and K. Broda. 2018. “Inductive Learning of Answer Set Programs from Noisy Examples.” *Advances in Cognitive Systems*, 7, 57–76.
- M. Law, A. Russo, and K. Broda. 2020. “The ILASP system for Inductive Learning of Answer Set Programs.” *arXiv preprint*, arXiv:2005.00904.
- M. Law, A. Russo, and K. Broda. 2015. *The ILASP System for Learning Answer Set Programs*. (2015). <https://www.ilasp.com>.
- B. G. León, M. Shanahan, and F. Belardinelli. 2022. “In a Nutshell, the Human Asked for This: Latent Goals for Following Temporal Specifications.” In: *International Conference on Learning Representations (ICLR)*.
- A. C. Li, T. Q. Klassen, A. Wang, P. A. Alamdari, and S. A. McIlraith. 2025. “Ground-Compose-Reinforce: Tasking Reinforcement Learning Agents through Formal Language.” In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- A. C. Li, Z. Chen, T. Q. Klassen, P. Vaezipoor, R. Toro Icarte, and S. A. McIlraith. 2024. “Reward Machines for Deep RL in Noisy and Uncertain Environments.” In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- A. C. Li, Z. Chen, P. Vaezipoor, T. Q. Klassen, R. Toro Icarte, and S. A. McIlraith. 2022. “Noisy Symbolic Abstractions for Deep RL: A Case Study with Reward Machines.” In: *Deep Reinforcement Learning Workshop at the Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. 2018. “RLlib: Abstractions for Distributed Reinforcement Learning.” In: *International Conference on Machine Learning (ICML)*.
- V. Lifschitz. 2019. *Answer Set Programming*. Vol. 3. Springer.
- T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2016. “Continuous Control with Deep Reinforcement Learning.” In: *International Conference on Learning Representations (ICLR)*.
- O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard. 2022. “Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks.” *IEEE Robotics and Automation Letters*, 7, 3, 7327–7334.
- S. Muggleton. 1991. “Inductive logic programming.” *New Generation Computing*, 8, 295–318.
- A. Y. Ng, D. Harada, and S. Russell. 1999. “Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping.” In: *International Conference on Machine Learning (ICML)*.
- L. Nodari. 2023. “Adversarial Attacks to Reward Machine-based Reinforcement Learning.” Master’s thesis. Università degli Studi di Brescia.
- L. Ouyang et al. 2022. “Training language models to follow instructions with human feedback.” In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*.

- R. Parac, L. Nodari, L. Ardon, D. Furelos-Blanco, F. Cerutti, and A. Russo. 2024. “Learning Robust Reward Machines from Noisy Labels.” In: *International Conference on Principles of Knowledge Representation and Reasoning (KR)*.
- W. Qiu, W. Mao, and H. Zhu. 2023. “Instructing Goal-Conditioned Reinforcement Learning Agents with Temporal Logic Objectives.” In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. 2021. “Stable-Baselines3: Reliable Reinforcement Learning Implementations.” *Journal of Machine Learning Research*, 22, 268, 1–8.
- J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. 2017. “Proximal Policy Optimization Algorithms.” *arXiv preprint*, arXiv:1707.06347.
- D. Silver et al. 2016. “Mastering the game of Go with deep neural networks and tree search.” *Nature*, 529, 7587, 484–489.
- R. S. Sutton and A. G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press.
- R. Toro Icarte, T. Q. Klassen, R. Valenzano, M. P. Castro, E. Waldie, and S. A. McIlraith. 2023. “Learning reward machines: A study in partially observable reinforcement learning.” *Artificial Intelligence*, 323, 103989.
- R. Toro Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. 2022. “Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning.” *Journal of Artificial Intelligence Research*, 73, 173–208.
- R. Toro Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. 2018. “Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning.” In: *International Conference on Machine Learning (ICML)*.
- R. Toro Icarte, E. Waldie, T. Q. Klassen, R. A. Valenzano, M. P. Castro, and S. A. McIlraith. 2019. “Learning Reward Machines for Partially Observable Reinforcement Learning.” In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- E. Umili, F. Argenziano, and R. Capobianco. 2024. “Neural Reward Machines.” In: *European Conference on Artificial Intelligence (ECAI)*.
- P. Vaezipoor, A. C. Li, R. Toro Icarte, and S. A. McIlraith. 2021. “LTL2Action: Generalizing LTL Instructions for Multi-Task RL.” In: *International Conference on Machine Learning (ICML)*.
- C. K. Verginis, C. Köprülü, S. Chinchali, and U. Topcu. 2024. “Joint Learning of Reward Machines and Policies in Environments with Partially Known Semantics.” *Artificial Intelligence*, 333, 104146.
- C. J. C. H. Watkins and P. Dayan. 1992. “Q-learning.” *Machine Learning*, 8, 3, 279–292.
- Z. Xu, I. Gavran, Y. Ahmad, R. Majumdar, D. Neider, U. Topcu, and B. Wu. 2020. “Joint Inference of Reward Machines and Policies for Reinforcement Learning.” In: *International Conference on Automated Planning and Scheduling (ICAPS)*.
- B. Yalcinkaya, N. Lauffer, M. Vazquez-Chanlatte, and S. Seshia. 2023. “Automata Conditioned Reinforcement Learning with Experience Replay.” In: *Goal-Conditioned Reinforcement Learning Workshop (GCRL) at the Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- B. Yalcinkaya, N. Lauffer, M. Vazquez-Chanlatte, and S. Seshia. 2024. “Compositional Automata Embeddings for Goal-Conditioned Reinforcement Learning.” In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- C. Yu, A. Velu, E. Vinitzky, J. Gao, Y. Wang, A. M. Bayen, and Y. Wu. 2022. “The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games.” In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*.

A Notation

Table 5 contains the notation used throughout the paper.

Table 5. Summary of mathematical notation.

Symbol	Description
Reinforcement Learning and (Noisy) Labelled MDPs (Sections 2.1 and 3)	
$\mathcal{S}$	Set of MDP states
s_0	Initial MDP state
$\mathcal{A}$	Set of actions
h	State-action history
p	Probability transition function
r	Reward function
τ	Termination function
τ^T	Terminal history indicator
τ^G	Goal history indicator
γ	Discount factor
π	Policy
$\mathcal{P}$	Set of propositions

Continued on next page...

Table 5 – continued

Symbol	Description
l	A single proposition
$\tilde{l}$	Noisy detection for proposition l
L	Label (a subset of propositions)
$\mathcal{L}$	Perfect labelling function
$\tilde{\mathcal{L}}$	Noisy labelling function
$\tilde{\mathcal{P}}$	Set of propositions detected by a noisy labelling function
λ	Label trace
$\tilde{\lambda}$	Noisy label trace
$[\tilde{\mathcal{L}}(s, a, s')]_l$	Probability of observing proposition l on transition $\langle s, a, s' \rangle$
Reward Machines (Section 2.2)	
$\mathcal{RM}$	Reward machine
U	Set of RM states
u_0	Initial RM state
u_A	Accepting RM state
u_R	Rejecting RM state
δ^u	RM state-transition function
δ^r	RM reward-transition function
Inductive Learning of Answer Set Programs (Section 2.3)	
B	Background knowledge of LAS task
S_M	Hypothesis space of a LAS task
E	Set of WCDPI examples of a LAS task
e_{id}	Identifier of a WCDPI example
e_{pen}	Penalty of a WCDPI example
$\langle e_{inc}, e_{exc} \rangle$	Partial interpretation of a WCDPI example
e_{ctx}	Context of a WCDPI example
H	Hypothesis (i.e., inductive solution of a LAS task)
PROB-IRM (Sections 4 to 6)	
λ'	Grounded trace (obtained by sampling from the noisy trace)
$n_{\lambda'}$	Number of grounded traces sampled from a noisy trace
φ	Logical transition function
$\hat{\varphi}$	Negation of the logical transition function
$U_{\max}$	Maximum set of states available to learn an RM
κ	Maximum number of edges from one state to another
w_c	Class-based weight multiplier for WCDPIs, inversely proportional to class frequency
w_{to}	Trace origin multiplier
t_{epen}	Minimum WCDPI penalty
$\tilde{u}$	RM state belief
$\tilde{\delta}^r$	Reward over RM state beliefs
β	RM relearning threshold

B Proof of Proposition 5.3

Recall from Definition 5.2 that the reward over RM state beliefs is:

$$\tilde{\delta}^r(\tilde{u}_t, \tilde{u}_{t+1}) = \sum_{\substack{u_t, u_{t+1} \in U, \\ L_t \in 2^P}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u_{t+1}] \delta^r(u_t, u_{t+1}), \quad (4)$$

where, under Assumption 3, the deterministic reward-transition function is:

$$\delta^r(u_t, u_{t+1}) = \begin{cases} 1 & \text{if } u_t \neq u_A \text{ and } u_{t+1} = u_A; \\ -1 & \text{if } u_t \neq u_R \text{ and } u_{t+1} = u_R; \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

The proof proceeds in three steps: first we expand the reward function using the deterministic reward structure, then we analyse how beliefs update for the accepting and rejecting states, and finally we combine these results to express the reward as a difference of belief updates.

Step 1: Expanding the Reward Function. We substitute $\delta^r(u_t, u_{t+1})$ into $\tilde{\delta}^r(\tilde{u}_t, \tilde{u}_{t+1})$. Since the reward is non-zero only when $u_t \notin \{u_A, u_R\}$ and $u_{t+1} \in \{u_A, u_R\}$, we can restrict the summation:

$$\begin{aligned} \tilde{\delta}^r(\tilde{u}_t, \tilde{u}_{t+1}) &= \sum_{\substack{u_t \in U \setminus \{u_A, u_R\}, \\ u_{t+1} = u_A, \\ L_t \in 2^{\mathcal{P}}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u_{t+1}] \cdot 1 \\ &+ \sum_{\substack{u_t \in U \setminus \{u_A, u_R\}, \\ u_{t+1} = u_R, \\ L_t \in 2^{\mathcal{P}}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u_{t+1}] \cdot (-1). \end{aligned}$$

Substituting the specific values $u_{t+1} = u_A$ and $u_{t+1} = u_R$ into their respective indicator functions:

$$\begin{aligned} \tilde{\delta}^r(\tilde{u}_t, \tilde{u}_{t+1}) &= \sum_{\substack{u_t \in U \setminus \{u_A, u_R\}, \\ L_t \in 2^{\mathcal{P}}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u_A] \\ &- \sum_{\substack{u_t \in U \setminus \{u_A, u_R\}, \\ L_t \in 2^{\mathcal{P}}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u_R]. \end{aligned} \quad (7)$$

Step 2: Belief Updates for Terminal States. We now analyse how the belief in the accepting state evolves. From the belief update equation in Definition 5.1:

$$\tilde{u}_{t+1}(u) = \sum_{\substack{u_t \in U, \\ L_t \in 2^{\mathcal{P}}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u]. \quad (3)$$

For the accepting state u_A , we partition the summation over u_t into three disjoint cases: non-terminal states, the accepting state itself, and the rejecting state:

$$\begin{aligned} \tilde{u}_{t+1}(u_A) &= \sum_{\substack{u_t \in U, \\ L_t \in 2^{\mathcal{P}}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u_A] \\ &= \sum_{\substack{u_t \in U \setminus \{u_A, u_R\}, \\ L_t \in 2^{\mathcal{P}}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u_A] \\ &+ \sum_{L_t \in 2^{\mathcal{P}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_A) \mathbf{1}[\delta^u(u_A, L_t) = u_A] \\ &+ 0. \quad (\text{no transition from } u_R \text{ to } u_A) \end{aligned} \quad (8)$$

The second term simplifies as follows:

$$\begin{aligned}
& \sum_{L_t \in 2^{\mathcal{P}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_A) \mathbf{1}[\delta^u(u_A, L_t) = u_A] \\
&= \tilde{u}_t(u_A) \sum_{L_t \in 2^{\mathcal{P}}} P(L_t \mid s_t, a_t, s_{t+1}) \mathbf{1}[\delta^u(u_A, L_t) = u_A] \\
&= \tilde{u}_t(u_A) \sum_{L_t \in 2^{\mathcal{P}}} P(L_t \mid s_t, a_t, s_{t+1}) \quad (\text{absorbing state: indicator always 1}) \\
&= \tilde{u}_t(u_A), \quad (\text{law of total probability})
\end{aligned}$$

where the second-to-last step uses the fact that $\delta^u(u_A, L_t) = u_A$ for all L_t , and the last step uses $\sum_{L_t \in 2^{\mathcal{P}}} P(L_t \mid s_t, a_t, s_{t+1}) = 1$.

Substituting this back into Equation (8) and rearranging:

$$\tilde{u}_{t+1}(u_A) - \tilde{u}_t(u_A) = \sum_{\substack{u_t \in U \setminus \{u_A, u_R\}, \\ L_t \in 2^{\mathcal{P}}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u_A]. \quad (9)$$

By analogous reasoning for the rejecting state u_R :

$$\tilde{u}_{t+1}(u_R) - \tilde{u}_t(u_R) = \sum_{\substack{u_t \in U \setminus \{u_A, u_R\}, \\ L_t \in 2^{\mathcal{P}}}} P(L_t \mid s_t, a_t, s_{t+1}) \tilde{u}_t(u_t) \mathbf{1}[\delta^u(u_t, L_t) = u_R]. \quad (10)$$

Step 3: Final Substitution. Comparing Equation (7) with Equations (9) and (10), we see that the two summations in the expression are exactly the belief differences. Therefore:

$$\tilde{\delta}^r(\tilde{u}_t, \tilde{u}_{t+1}) = (\tilde{u}_{t+1}(u_A) - \tilde{u}_t(u_A)) - (\tilde{u}_{t+1}(u_R) - \tilde{u}_t(u_R)),$$

which completes the proof.

Received 15 April 2026; accepted 01 June 2026