Generating Explainable Counterfactual Policies through Temporal Logic Queries

Arnaud Lequen[1] , **Clément Legrand-Lixon**[2] , **Léo Saulières**[3]
[1]Linköping University, Sweden
[2]Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
[3]Univ. Toulouse, INRAE-MIAT, Toulouse, France
arnaud.lequen@liu.se, clement.legrand@univ-lille.fr, leo.saulieres@inrae.fr

Abstract

As reinforcement learning (RL) agents are deployed in increasingly complex environments, ensuring that their behavior complies with the user's needs has become a central challenge in eXplainable RL (XRL). An agent's policy may solve a given problem, but some of its choices can seem counter-intuitive or surprising to the user, who may have wished to see the agent accomplish its goal in a different way, and may wonder: what if the agent acted with a different intent in mind? Scenarios that answer this question are called counterfactual policies. In this work, we propose a framework that allows the user to request these alternative policies by formulating preferences about the behavior of the agent. These preferences are expressed in Linear Temporal Logic on finite traces (LTL_f), a formal yet intuitive language that allows reasoning about deterministic sequences of actions. We synthesize the corresponding counterfactual policies using a multi-objective reinforcement learning algorithm, which produces a diverse set of alternative strategies balancing the agent's original policy with the one envisioned by the user. By comparing these strategies, our framework sheds light on the rationale behind the agent's decisions. Experimental trials show that such a set of policies can be synthesized in reasonable time.

1 Introduction

In the context of sequential decision-making, Reinforcement Learning (RL) agents often achieve strong performance, yet the objectives that shaped their learned policies may remain opaque, especially when the underlying reward function is unknown. In such cases, users may observe behaviors that comply with the intended objective of the agent, but do not fully match the users' preferences. The resulting policy is thus hard to understand, and the user has no clear way to adapt the agent's behavior to his or her needs. This challenge lies at the intersection of Reinforcement Learning and eXplainable AI (XAI), and motivates the need for tools that allow users to question and explore alternative behaviors of an RL agent.

We address this limitation by proposing a framework for generating counterfactual policies guided by user-specified preferences, expressed in Linear Temporal Logic on finite traces (LTL_f). LTL_f is a modal logic that expresses properties about sequences of states and is more human-readable than reward functions. This makes preference design less error-prone, more likely to lead to a desired outcome, and more interpretable. In this paper, we tackle the problem of altering an input policy π so that it complies as well as possible with user preferences.

Producing a single modified policy may not be satisfying, since that policy may turn out different from what the user envisioned. Instead, we propose to produce a *set* of counterfactual policies that trade off fidelity to π and satisfaction of the user's preferences, to varying degrees. This naturally leads to a multi-objective formulation: learning a new policy that is as close as possible to the reference policy while complying with temporal preferences.

Scalarizing the objectives into a single reward would require fixing arbitrary weights and would yield only one compromise, which gives the user few clues to understand π. Instead, we formulate the problem as a multi-objective RL task and adopt a multi-policy approach, allowing us to approximate the set of dominating policies and produce diverse counterfactual policies in a single training phase. In addition, a strength of our approach is that it is completely agnostic to the origin of the policy: we do not assume access to any information used during training, such as Q-values.

To achieve this, we encode both the reference policy and the temporal logic preferences as reward machines, which are structured reward signals. We then employ a variant of Pareto Q-Learning adapted to reward machines to learn policies among the most relevant ones, along the Pareto front. We evaluate the proposed framework on two illustrative environments, demonstrating its ability to generate meaningful and interpretable counterfactual behaviors.

The remainder of the paper is organized as follows. Section 2 provides the necessary background. In Section 3, we define the *Counterfactual Policies through Temporal Preferences* problem. Section 4 presents our approach. Section 5 reports experimental results. Section 6 discusses key insights and limitations. Section 7 reviews related work. Finally, Section 8 concludes.

2 Background

We begin by introducing the formalism behind reinforcement learning, the multi-objective context, its interactions with reward machines, as well as LTL_f.

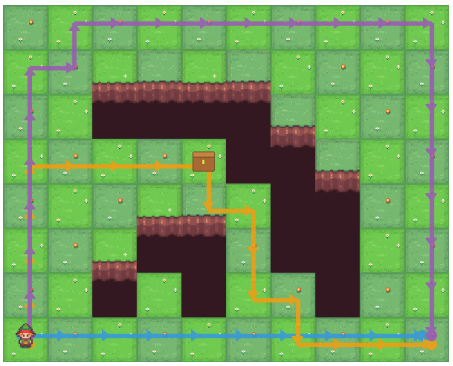


Figure 1: A Cliffwalking problem. The agent wishes to move to the bottom-right hand corner of the map. The shortest path (the reference policy) is in blue, while the yellow path also takes into account the user's preference to collect the treasure, and the purple path to stay away from cliffs.

2.1 Multi-Objective Reinforcement Learning

Labelled Transition System A *transition system* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathsf{succ} \rangle$ where $\mathcal{S}$ is a set of symbols called *states*, and $\mathcal{A}$ a set of symbols called *actions*. Each state s is associated with $A(s) \subseteq \mathcal{A}$, the set of actions that are *applicable* in s. $\mathsf{succ} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the *successor* partial function, which associates to each pair (s, a) of state and action such that $a \in A(s)$, its successor state $s' = \mathsf{succ}(s, a)$. s' is the result of the application of a in s.

Given a set of propositional symbols $\mathcal{P}$, a *labelled transition system* $\langle \mathcal{S}, \mathcal{A}, \mathsf{succ}, L \rangle$ is a state space enriched with a *labelling function* $L : \mathcal{S} \to 2^{\mathcal{P}}$. This function indicates which propositions are true in each state. For $p \in \mathcal{P}$, when $p \in L(s)$, we will often write, by a slight abuse of notation, $p \in s$.

Intuitively, a transition system represents the environment in which an agent evolves, as well as its dynamics. Consider for example Figure 1. The states are the different positions of the agent, and the actions are `UP`, `DOWN`, `LEFT`, and `RIGHT`. In a state, applicable actions are those that do not move the agent out of the map. Applying an action in a cell moves the agent in the direction indicated by that action, except if the agent previously fell into a cliff, in which case it is stuck. Some cells are states labelled with propositions, such as `treasure` or `cliff`. Cells that are not next to a cliff are labelled `safe`.

Markov Decision Process A *Markov Decision Process* (MDP) (Sutton and Barto 2018) is a tuple $M = \langle \mathcal{S}, \mathcal{A}, R, p, \gamma \rangle$. $\mathcal{S}$ and $\mathcal{A}$ are the states and actions sets. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ denotes the reward function, and $R(s, a, s')$ is the reward obtained by performing action a from state s and reaching s'. $\gamma \in (0, 1]$ is the discount factor. $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ denotes the transition function: $p(s, a, s')$ is the probability of reaching the state s' by performing action a from state s. In this paper, however, we focus on deterministic MDPs, where $p(s, a, s') \in \{0, 1\}$: transitions are deterministic. Slightly abusing the notations, we will thus consider (labelled) MDPs of the form $M = \langle \mathcal{S}, \mathcal{A}, R, \mathsf{succ}, \gamma \rangle$, where $\langle \mathcal{S}, \mathcal{A}, \mathsf{succ}, L \rangle$ is a labelled transition system.

In general, solutions to an MDP take the form of *policies*. A policy π is a mapping $\pi : \mathcal{S}^+ \to \mathcal{A}$ (where $\mathcal{S}^+$ are tuples of size at least 1 of states) that associates the history of visited states with the action that the agent chooses. For a given state sequence $s_0, \ldots, s_t$, $\pi(s_0, \ldots, s_t) = a_t$ is the action chosen by the agent at step t, leading to state $\mathsf{succ}(s_t, a_t) = s_{t+1}$. In this paper, for a given initial state $s_0 \in \mathcal{S}$, we will denote $\pi^*[s_0] = \langle s_0, s_1, \ldots, s_n \rangle$ the *trace* of states resulting from the unfolding of the policy π on that state up to horizon n.

In general, the solution to an MDP is a policy that maximizes expected discounted return at horizon n. In the deterministic setting, this is expressed as follows:

$$V_n^\pi = \sum_{t=0}^{n} \gamma^t r_{t+1}^\pi$$

where r_t is the reward obtained at step t, after unfolding the policy π, i.e. $r_{t+1}^\pi = R(s_t, a_t, s_{t+1})$. In our setting, we will only consider *bounded* executions of policies with up to n steps, i.e. $n < \infty$.

A policy is *stationary* if it depends only on the current state (Markov policy), i.e., there exists $\tilde{\pi} : \mathcal{S} \to \mathcal{A}$ such that for every reachable history $h_t \in \mathcal{S}^+$ ending in state s_t, $\pi(h_t) = \tilde{\pi}(s_t)$. Otherwise, it is *non-stationary*.

In Figure 1, a reward function that encourages reaching the treasure can be defined as follows:

$$R(s, a, s') = \begin{cases} 1 & \text{if } \texttt{treasure} \in L(s'), \\ 0 & \text{otherwise.} \end{cases}$$

A policy that solves the associated MDP is one that leads the agent to the treasure through a shortest path.

Multi-Objective Reinforcement Learning A multi-objective MDP (MOMDP) (Wiering and de Jong 2007) is a tuple $M_{MO} = \langle \mathcal{S}, \mathcal{A}, \mathbf{R}, p, \gamma \rangle$ where $\mathcal{S}, \mathcal{A}, p$ and γ are similar to MDP, and $\mathbf{R}$ is a *multi-objective reward function*. $\mathbf{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^d$ is a function that returns a vector composed of d scalars, each representing the reward associated with an objective (for $d \geq 2$). Since there is no canonical order in $\mathbb{R}^d$, solutions to the MOMDP are defined by means of *Pareto-optimal* policies.

Let Π be the set of all policies for an MOMDP M. The Pareto front of M is the non-dominated set (Roijers et al. 2013):

$$PF(\Pi) = \{\pi \in \Pi | \quad \nexists \pi' \in \Pi : \mathbf{V}^{\pi'} \succ_P \mathbf{V}^\pi\}$$

where $\mathbf{V}^\pi \in \mathbb{R}^d$ is the value function associated with policy π, defined as $\mathbf{V}^\pi = \sum_{t=0}^{\infty} \gamma^t \mathbf{r}_{t+1}^\pi$. $\mathbf{r}_t^\pi$ is the reward obtained at time step t after unfolding π. The Pareto dominance relation $\succ_P$ is such that

$$\mathbf{V}^\pi \succ_P \mathbf{V}^{\pi'} \text{ iff } (\forall i : \mathbf{V}_i^\pi \geq \mathbf{V}_i^{\pi'}) \wedge (\exists i : \mathbf{V}_i^\pi > \mathbf{V}_i^{\pi'})$$

where $\mathbf{V}_i^\pi$ is the i-th component of the value function of a policy π. The Pareto front contains the set of *Pareto-optimal*

policies that are not Pareto dominated in the sense of $\succ_P$, i.e., policies for which there is no other policy with equal or greater value for all objectives.

Work in this field can be divided into two groups of algorithms (Hayes et al. 2022) depending on whether or not a clear hierarchy regarding the objectives to be achieved is known: when they is known, single policy algorithms scalarize the reward function to learn a unique policy aligned with preferences, using techniques akin to single-objective reinforcement learning. When preferences cannot be quantified, multi-policy algorithms are used to obtain a set of policies that maximize the various objectives in different proportions, in the sense that they are Pareto-optimal. An example of such a multi-policy algorithm is Pareto-Q-learning (PQL) (Moffaert and Nowé 2014) which is a multi-objective adaptation of Q-learning (Watkins and Dayan 1992).

The hypervolume (Bader and Zitzler 2011) is a common metric used to assess the quality and the convergence of a Pareto front. It represents the volume of the region dominated by the Pareto front, and must be computed with a carefully chosen reference point, delimiting the region. In practice, the point whose coordinates represent the lowest (in a maximization context) possible values for each objective is considered (i.e., the *nadir* point).

2.2 Reward Machines

Deterministic Finite Automata A Deterministic Finite Automaton (DFA) (Hopcroft, Motwani, and Ullman 2001) over an alphabet Σ of input symbols is a tuple $D = \langle U, \delta, u_0, F\rangle$ where U is the set of states, $\delta : U \times \Sigma \to U$ is the deterministic transition function, u_0 is the initial state, and F is the set of accepting states. Given a finite word $x = \sigma_1\sigma_2\cdots\sigma_n \in \Sigma^*$, let $\delta^* : U\times\Sigma^* \to U$ be the extended transition function defined by $\delta^*(u, \varepsilon) = u$ (where ε is the empty word) and, for all $\sigma \in \Sigma$, $\delta^*(u, \sigma x) = \delta^*(\delta(u, \sigma), x)$. The word x is accepted by D iff $\delta^*(u_0, x) \in F$.

Reward Machine A Reward Machine (RM) is a finite state machine that selects a reward function depending on its current state (Icarte et al. 2018), and given a state and an action, outputs a scalar reward. Given a set of propositional symbols $\mathcal{P}$, a set of states $\mathcal{S}$ and a set of actions $\mathcal{A}$, an RM is defined as a tuple $R = \langle U, u_0, \delta_s, \delta_r\rangle$. U and $u_0 \in U$ denote, respectively, a finite set of states and an initial state. $\delta_s : U \times 2^{\mathcal{P}} \to U$ denotes the state-transition function; $\delta_s(u, \sigma)$ is the state reached after receiving a set $\sigma \in 2^{\mathcal{P}}$ of propositions and being in u. $\delta_r : U\times U \to [\mathcal{S}\times\mathcal{A}\times\mathcal{S} \to \mathbb{R}]$ denotes the reward-transition function; $\delta_r(u, u')$ is the reward function to use when transitioning from state u to u' of the RM.

As an illustration, Figure 2 shows a reward machine that encourages the agent to not linger next to a cliff for too long.

MORL with Reward Machines Extending reward machines to MORL requires considering one RM per objective. Thus, the Multi-Objective Reward Machine (MORM) formalism (Aronis 2025; Lequen, Legrand-Lixon, and Saulières 2026a) proposes representing a reward function

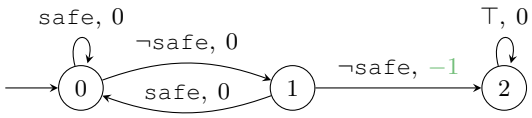


Figure 2: Reward machine that gives a reward of -1 the first time the agent is in two consecutive states not labelled `safe`.

$\mathbf{R}$ as a tuple of d reward machines. Formally, we have $\mathbf{R} = \langle \mathbf{U}, \mathbf{u}_0, \boldsymbol{\delta}_s, \boldsymbol{\delta}_r\rangle$ where $\mathbf{U} = U_1 \times \cdots \times U_d$ is the set of joint states of the d RMs, $\mathbf{u}_0$ is the joint initial state, $\boldsymbol{\delta}_s$ and $\boldsymbol{\delta}_r$ are respectively the joint transition functions of states and reward functions of the d RMs. In order to solve MORM problems, we use a variant of PQL (Moffaert and Nowé 2014) adapted to support reward machines.

2.3 Linear Temporal Logic on Finite Traces (LTL_f)

Linear Temporal Logic on finite traces (LTL_f) is a formalism for expressing properties over finite sequences of states, called *traces*.

Syntax Given a set $\mathcal{P}$ of propositional symbols, formulas of LTL_f are defined inductively as:

$$\varphi := \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi \,\mathsf{U}\, \varphi$$

where $p \in \mathcal{P}$ is a propositional symbol, $\bigcirc$ is the *next* operator, and U is the *until* operator. We use the following standard abbreviations: $\bot = \neg\top$ (false), $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$ (disjunction), $\varphi \Rightarrow \psi = \neg\varphi \vee \psi$ (implication), $\Diamond\varphi = \top \,\mathsf{U}\, \varphi$ (eventually), and $\Box\varphi = \neg\Diamond\neg\varphi$ (always/globally).

Intuitively, $\bigcirc\varphi$ states that φ must hold at the next step, $\Diamond\varphi$ states that φ must hold at some future step, and $\Box\varphi$ states that φ must hold at every step of the trace. The until operator $\varphi \,\mathsf{U}\, \psi$ means that φ must keep holding until ψ becomes true. As an example, $\Box(\neg\texttt{safe} \Rightarrow \Diamond\texttt{safe})$ expresses that whenever the agent is on a dangerous cell ($\neg\texttt{safe}$), it must come back eventually to a safe cell.

Semantics A *world* w over $\mathcal{P}$ is then represented as a set $w \subseteq \mathcal{P}$ of the symbols that are true in w. If $p \notin w$, then p is false in w. LTL_f (Giacomo and Vardi 2013) defines semantics for finite world sequences. A trace t is a finite sequence of worlds, with $t = \langle w_1, \ldots, w_n\rangle \in (2^{\mathcal{P}})^*$.

$$
\begin{aligned}
&t, i \models p && \text{iff } p \in w_i\\
&t, i \models \neg\varphi && \text{iff } t, i \not\models \varphi\\
&t, i \models \varphi_1 \wedge \varphi_2 && \text{iff } t, i \models \varphi_1 \text{ and } t, i \models \varphi_2\\
&t, i \models \bigcirc\varphi && \text{iff } i < n \text{ and } t, (i+1) \models \varphi\\
&t, i \models \varphi_1 \mathsf{U} \varphi_2 && \text{iff } \exists j \in [\![i, n]\!] \text{ s.t. } t, j \models \varphi_2\\
& && \quad \text{and } \forall k \in [\![i, j-1]\!], \text{ we have } t, k \models \varphi_1
\end{aligned}
$$

If $t, 0 \models \varphi$, we say that φ is true in t or that t satisfies φ, written $t \models \varphi$. We use $[\![i, j]\!]$ as a shorthand for the set of integers $\{i, i+1, \ldots, j\}$. In practice, in our setting, a world is the labelling of a state.

Safety and Co-Safety Let $u \preceq t$ denote that u is a prefix of a trace t. A formula φ is a *safety* property if every counterexample has a finite bad prefix u, i.e., a prefix that can never be extended into a trace t' satisfying φ:

$$\forall t \in (2^{\mathcal{P}})^*, \text{if } t \not\models \varphi, \text{then } \exists u \preceq t \;\; \forall t' \succeq u, \; t' \not\models \varphi.$$

Intuitively, once a bad prefix has occurred, no continuation can repair the violation. A typical example is $\square\,\texttt{safe}$.

Dually, φ is *co-safe* if every satisfying trace has a finite good prefix, i.e.,

$$\forall t \in (2^{\mathcal{P}})^*, \text{if } t \models \varphi, \text{then } \exists u \preceq t \;\; \forall t' \succeq u, \; t' \models \varphi.$$

Intuitively, once such a good prefix is observed, satisfaction is guaranteed for any continuation. A typical example is $\Diamond\,\texttt{goal}$.

There exist efficient algorithms for checking whether an LTL_f formula is a safety (resp. co-safety) property or not (Latvala 2003). They essentially consist in building the associated finite state automaton, and performing a reachability analysis on (a variant of) it.

Not all properties are safety or co-safety properties. In the rest of this paper, we will only consider LTL_f formulas that are safe or co-safe.

3 Counterfactual Policies through Temporal Preferences

In this section, we introduce the problem that we address in the rest of this paper.

Suppose that a policy π is available for a problem modelled by an MDP, but where the reward function that led to its acquisition is unknown. In other words, although the agent's behavior is observable, the exact objective it optimises remains unclear. In this context, a user can write a formula φ (or a set of formulas $\{\varphi_i\}_{i\leq N}$) that expresses preferences: what would happen if the agent also had to take into account an additional objective, while maintaining its current behavior as a reference point?

These preferences, expressed as temporal logic formulas, are meant to perturb the original policy so that it deviates from its original objective as little as possible, while complying with the preferences expressed by the user. The aim is then to generate counterfactual policies that reflect different trade-offs between the original behavior and the satisfaction of the desired properties.

Counterfactual Policies Let π_1 and π_2 be two policies on $\mathcal{T} = \langle \mathcal{S}, \mathcal{A}, \mathsf{succ}, L\rangle$. The distance between them is $d(\pi_1, \pi_2) = |\{s \in \mathcal{S} \mid \pi_1(s) \neq \pi_2(s)\}|$. Let $\Psi = \{\varphi_1, \ldots, \varphi_N\}$ be a set of LTL formulas on the set of propositions $\mathcal{P}$ underlying $\mathcal{T}$. The set of policies that are counterfactual to π with respect to Ψ are the ones which comply with Ψ while deviating the least from π:

$$\mathcal{C}^{\pi}_{\Psi} = \text{argmin}_{\pi_c}\{d(\pi_c, \pi) \mid \forall \varphi_i \in \Psi, \forall s \in \mathcal{S}, \pi_c^*[s] \models \varphi_i\}.$$

Given Ψ and π, in what follows, let us denote Π a set of policies that are counterfactual to π w.r.t. a subset of the formulas of Ψ:

$$\Pi \subseteq \bigcup_{\Phi \subseteq \Psi} \mathcal{C}^{\pi}_{\Phi}$$

Π is *consistent* if, for $\pi_1, \pi_2 \in \Pi$, s. t. $\pi_1 \neq \pi_2$, if $\pi_1 \in \mathcal{C}^{\pi}_{\Phi_1}$ and $\pi_2 \in \mathcal{C}^{\pi}_{\Phi_2}$ with $\Phi_1 \subseteq \Phi_2 \subseteq \Psi$, then $d(\pi_1, \pi) < d(\pi_2, \pi)$. In other words, if some π_1 complies with fewer objectives than π_2, then it must respect more closely the original policy. Note that this forbids two policies from a single $\mathcal{C}^{\pi}_{\Phi}$ from belonging to Π simultaneously.

If Π is consistent, then it is *saturated* if there exists no counterfactual policy π' with respect to (a subset of) Ψ such that adding π' to Π makes it remain consistent.

Note that such a consistent, saturated set of counterfactual policies is akin to a Pareto front: we exclude counterfactual policies that are less interesting than others because they comply with fewer objectives or do not comply as well with the reference policy.

Problem Definition The Counterfactual Policies through Temporal Logic preferences (CFP-TL) problem takes as input a policy π and a set of N temporal logic formulas $\Psi = \{\varphi_1, \ldots, \varphi_N\}$ (with the assumption that they are individually either safe or co-safe), and outputs a set of counterfactual policies Π_{CFP} that satisfy, to varying degrees, the objectives expressed by these formulas, while remaining grounded in the original behavior.

Problem 1. CFP-TL: *Counterfactual Policies through Temporal Logic preferences problem*

Input: *A labeled transition system* $\mathcal{T}$
a policy π *on* $\mathcal{T}$
$\Psi = \{\varphi_i\}_{i\leq N}$ *a set of* LTL_f *formulas that are safe or co-safe*

Output: Π_{CFP} *a consistent, saturated set of policies that are counterfactual to* π *w.r.t. the subsets of* Ψ.

4 Method

In this section, we introduce our framework for learning policies that comply as well as possible with the properties that the user would like to see enforced, while still staying as close as possible to the original policy provided in input.

We approximate a solution to the *CFP-TL* problem by compiling it into a multi-objective MDP with reward machines, and then passing it to a MORL with RM algorithm. In practice, our method consists of two steps:

1. Generate reward machines $\mathbf{R} = (R_i)_{i\leq N+1}$ that model the policy π and user preferences $\{\varphi_i\}_{i\leq N}$, and create an MOMDP $M_C = \langle \mathcal{S}, \mathcal{A}, \mathbf{R}, \mathsf{succ}, \gamma\rangle$.
2. Solve M_C using a multi-policy algorithm.

This allows us to learn counterfactual policies that balance the various objectives represented by these reward machines. Since these policies are meant to deviate as little as possible from the input policy, we also propose an extension that summarizes the key differences with the original policy, making the interpretation and comparison of these policies clear to the user.

In this section, we show how to obtain the reward machines $\mathbf{R}$, discuss our choice of MORL with RM algorithm, and then show our method for summarizing policies.

4.1 Reward Machine Computation

Given a set $\{\varphi_i\}_{i\leq N}$ of user-specified formulas, and the input policy π, we build N reward machines $(R_{\varphi_i})_{i\leq N}$, that reward the agent for complying with the user's specified preferences. We also build a *policy reward machine* R_π, that rewards the agent for following the original policy. In the rest of this section, we show how to compute these reward machines, depending on the properties expressed by the formulas.

Converting Co-Safety Properties into Reward Machines Given a co-safety property φ, it can be converted into a DFA $D_\varphi = \langle U, \delta, u_0, F\rangle$ that accepts exactly the finite traces that satisfy φ (Giacomo and Vardi 2013), and is never blocked (i.e. there is always a transition that can be taken). We start by performing this step, such that the transitions of D_φ are conjunctions of propositional symbols. All that is left now is to allocate reward functions to each state.

Checking that φ is a co-safety property is done by computing the set of states $U_R \subseteq U$ from which there exists a path to a rejecting state. If the set $Reach(u_0)$ of states reachable from the initial state u_0 is such that $(Reach(u_0) \cap F) \setminus (U \setminus U_R) = \varnothing$, then φ is a co-safety property. Indeed, there is no non-accepting state that can be reached from a (reachable) accepting state.

In other words, once an accepting state has been reached, the run remains in an accepting state, and we know that the input will be accepted. We use this property to preemptively declare the co-safety property φ satisfied during the execution of a policy, without waiting for the end of the trace.

This means that we merge all accepting states into a single one u_f, and assign a reward of 1 to transitions to u_f, and 0 to the others.

Converting Safety Properties into Reward Machines Safety properties are the dual of co-safety properties, in the sense that instead of detecting accepting states that cannot be left (for a rejecting state), we detect rejecting states that cannot be left (for an accepting state). We then merge these states into a single one, creating a sink. Transitions leading to this sink yields a reward of -1, all other transitions 0.

Figure 2 shows the reward machine generated by the safety property $\square(\neg\texttt{safe} \wedge \bigcirc\neg\texttt{safe})$. Accepting states are states 0 and 1, and 2 is a sink. As a consequence, only the transition from 1 to 2 has a non-zero reward.

Policy Reward The calculation of R_π is straightforward based on π. The objective is to propose a reward machine that represents the agent's behavior, i.e. π. As the policy is deterministic, for each state $s \in \mathcal{S}$, the agent returns a single action, namely $\pi(s)$. Having access to π, we simply propose to provide a reward of 1 if the chosen action corresponds to that of π, and of $-c$ otherwise, where $c \in \mathbb{N}$. Formally, we have a reward machine with a single state, and thus a single reward function:

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A} \quad R_\pi(s,a) = \begin{cases} +1, & \text{if } a = \pi(s). \\ -c, & \text{otherwise.} \end{cases}$$

After this step, we are left with the multi-objective reward machines $\mathbf{R} = \{R_\pi, R_{\varphi_1}, \ldots, R_{\varphi_N}\}$, and a MOMDP $M_C = \langle \mathcal{S}, \mathcal{A}, \mathbf{R}, \mathsf{succ}, \gamma\rangle$.

4.2 Solving the MOMDP

The second step of the framework consists in learning a set of policies that solve MOMDP M_C, and that are solutions to our initial problem. Here, the agent must balance two different objectives: preserving the behavior induced by the original policy π (via R_π) and satisfying the preferences expressed by the temporal formula (via $(R_{\varphi_i})_{i\leq N}$).

Single- and Multi-Policy Algorithms A single-policy algorithm, based on the optimization of a single scalar reward obtained by a linear combination of the rewards of the rewards machines, is not suitable in this context. Indeed, in our problem, the weights that the user would like to attribute to each preference are unknown, maybe even to the user themselves, who is specifically seeking to explore different counterfactual trade-offs. Imposing an arbitrary weighting would therefore lead to learning a single policy, corresponding to a particular trade-off which might not suit the user well enough. This would lead to a phase of adjusting the weights until a suitable policy is found.

This setting naturally fits within the Multi-Objective Reinforcement Learning (MORL) paradigm. Among existing approaches, multi-policy algorithms are particularly well suited to our objective: they enable a single training session to learn a set of policies approximating the Pareto front, each corresponding to a different proportion of compliance between R_π and R_φ. Such an approach directly provides a spectrum of counterfactual policies, which the user can freely explore.

In this context, we use a variant of Pareto Q-Learning (PQL) that was adapted to work with reward machines (Aronis 2025). This multi-policy algorithm, based on Q-learning, allows us to learn non-stationary policies located on the Pareto front while exploiting the structure of RMs.

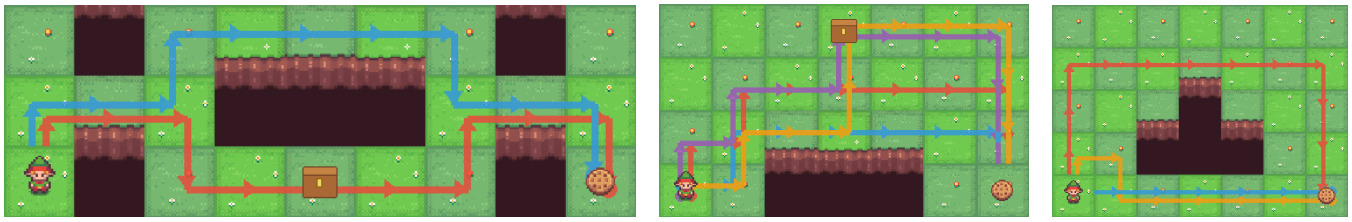
4.3 Summarizing and Comparing Policies

Our framework produces a set of counterfactual policies that comply, in varying proportions, with both the agent's original policy and the preference(s) specified by the user. Since these policies can be lengthy, they may be deemed hard to interpret. However, we can exploit the fact that these policies are close to π, provided in input, which we assume is known to the user.

It is easy to generate the set of actions of an output policy $\pi' \in \Pi_{\text{CFP}}$, since one only has to track the reward signal of the RM R_π to know where π' deviated. Showing these transitions to the user serves as a summary of the differences between π and a candidate policy π'.

5 Experimental Results

Experimental Setup All experiments were conducted with an AMD Ryzen 7 PRO 5850U processor. We constrained memory usage to at most 8 GB, and each run took a maximum time of 2 minutes to complete.



(a) Results of experiment 1. (b) Results of experiment 2. (c) Results of experiment 4.

Figure 3: Three Cliffwalking maps used in experiments and their associated results. For each experiment, the blue trajectory is the reference policy and all policies are Pareto optimal. For experiment 1 (left), a new policy satisfies the preference $\Diamond$ `treasure` (red). For experiment 2 (center), the policies satisfy the following preferences: $\Diamond$ `treasure` (yellow), $\Box$`safe` (red), and $\Diamond$ `treasure` $\wedge$ $\Box$ `safe` (purple). For experiment 4 (right), two new policies are generated using temporally-extended preferences: $\varphi_1 = \Diamond(\texttt{safe} \wedge \bigcirc\neg\texttt{safe} \wedge \bigcirc\bigcirc \texttt{safe})$ (yellow) and $\varphi_2 = \Box(\neg\texttt{safe} \Rightarrow \bigcirc\texttt{safe})$ (red).

Our implementation was done in Python 3.10, relying on the Gymnasium library (Towers et al. 2024). To compile input LTL$_f$ formulas into the automata used by our pipeline, we used Lydia (Zhu and Favorito 2025). Our code and results are available online (Lequen, Legrand-Lixon, and Saulières 2026b).

Set of Benchmarks Two deterministic domains are considered for our experiments. Cliffwalking, a classic gridworld environment introduced by (Sutton and Barto 2018), where an agent navigates to reach a defined goal on the map, avoiding cliffs. We also consider the possibility of the agent collecting a treasure, which is on some cells. Four maps are considered, presented in Figure 1, Figure 3, Figure 3b, and in Figure 3c. Each map is associated with its own user-defined preferences, described in Section 5.1.

Restaurant is another gridworld environment we propose to assess the performance of our algorithm. In this domain, a waiter has to serve food to two tables. They become dirty after 5 units of time, and the waiter has to clean them by throwing leftovers in the trash. The map is shown in Figure 6, and we consider two user-defined preferences on this map, presented in Section 5.2.

Baseline As a baseline, we use a standard Q-learning algorithm (Watkins and Dayan 1992) to show that it is more efficient to modify the policy on the fly using our method than to retrain a new policy meeting the preference of the user. Moreover, the MORL algorithm used by our method is itself an adaptation of Q-learning to our specific setting. For each domain, we compare the convergence rate of the algorithms for each experiment, adapting the reward functions (and environment if need be) passed to the Q-learning algorithm.

Parameters We set $c = 5$ in R_π, ensuring that the agent stays as close as possible to the reference policy.

For Cliffwalking, the maps contain between 54 and 130 states, we thus allowed 2×10^4 steps for the training of the agent, with $\gamma = 0.95$, an ϵ-greedy strategy to choose the action to perform in the current state ($\epsilon = 1.0$ in the beginning to favor exploration then linearly decreases at each step until it reaches 0.1 after 1200 steps, to favor exploitation). Each episode was limited to 30 steps.

For Restaurant, since the map contains 576 states, we allowed 4×10^4 timesteps, with $\gamma = 0.99$, with an ϵ-greedy strategy starting at $\epsilon = 1$ and reaching $\epsilon = 0.05$ after 1000 (resp. 15×10^3) steps for experiment 1 (resp. experiment 2). Again, each episode was limited to 30 steps.

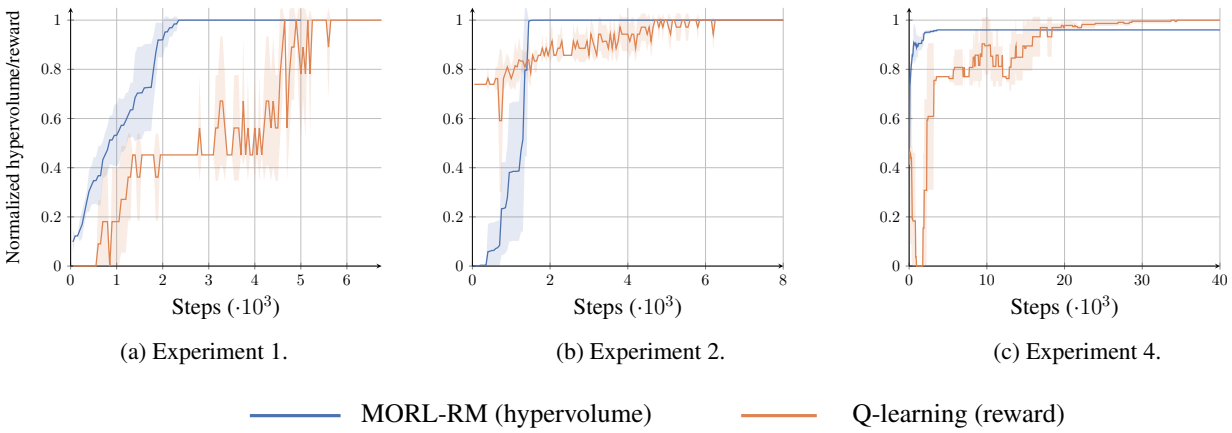
5.1 Cliffwalking

Episode During an episode, the agent navigates on a rectangular grid, starting from the bottom-left corner and aiming to reach a *goal* located at the bottom-right corner. The grid contains *cliff* cells spread across the map, which end the episode when the agent walks onto them. In addition, we introduce a special cell, labeled with `treasure`. In the original setting, the agent's state is its position. At each step, the agent can move in one of the four cardinal directions.

Reference Policy The reference policy is defined on every cell and follows the shortest path to reach the goal, avoiding cliffs without considering the treasure (e.g. blue path in Figure 3a). When several actions lead to the same solution, action `RIGHT` is preferred, then `DOWN`, then `LEFT`.

Experiment 1: The Other Side The first experiment is performed on the map in Figure 3a, which also presents the resulting policies. On this map, the reference policy follows the upper path, ignoring the treasure. In this situation, the preference expressed by the user is $\Diamond$ `treasure`, meaning that the agent eventually reaches the treasure. In addition to the reference policy (in blue), a policy satisfying the preference of the user is learned by our approach, thus matching our expectations. Note that the two policies are Pareto optimal, since any other trajectory would either deviate more than once from the reference policy, or take more steps to reach the treasure and/or the goal.

Experiment 2: Staying Safe We perform a second experiment on the map of Figure 3b, where the reference policy walks along cliffs at the bottom, disregarding the treasure



(a) Experiment 1. (b) Experiment 2. (c) Experiment 4.

Figure 4: Evolution of the hypervolume/reward per training step, for experiments 1 (left), 2 (center) and 4 (right) on Cliffwalking. Our method (MORL-RM) tracks the hypervolume of our set of policies, and Q-learning tracks the reward of the greedy policy. Note that, when the convergence was faster, we cut the graphs short to make them easier to read.

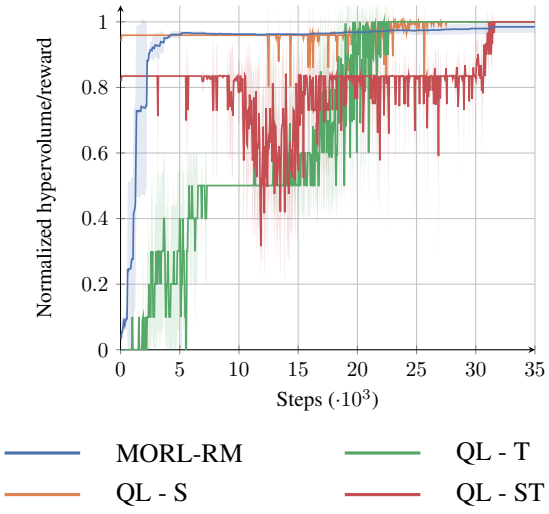


Figure 5: Evolution of the hypervolume/reward per training step for experiment 3 on Cliffwalking. For our method (MORL-RM), we track the hypervolume of the set of policies found. The Q-learning is applied with different reward functions: either rewarding safe behavior (QL-S), collecting the treasure (QL-T), or both (QL-ST).

located at the top of the grid and only focused on the shortest path. Here, two preferences are expressed by the user: $\Diamond$ `treasure` (same as experiment 1), and $\Box$ `safe`, meaning that the agent is always on a *safe* cell (which are cells that are at distance at least one from a cliff). Our algorithm is able to produce four different policies: the reference one, one for each preference of the user (in yellow and red), and one satisfying both preferences of the user (in purple), but deviating the most from the reference policy.

Experiment 3: Unsafe Treasure A third experiment is performed on the map presented in Figure 1, where the treasure is located at an unsafe cell. As in the previous experiment, the user specifies two preferences $\Diamond$ `treasure` and $\Box$ `safe`, however the treasure being unsafe, it will not be possible to obtain a policy satisfying simultaneously both preferences of the user. As expected, our algorithm returns three policies (see Figure 1): the reference one (in blue), and one for each preference specified by the user.

Experiment 4: Fear of the Void Our last experiment concerns a map where cliffs are located at its center. The user expresses the two following preferences for something that they wish to see the agent perform:

$$\varphi_1 = \Diamond(\texttt{safe} \wedge \bigcirc\neg\texttt{safe} \wedge \bigcirc\bigcirc\,\texttt{safe})$$
$$\varphi_2 = \Box(\neg\texttt{safe} \Rightarrow \bigcirc\texttt{safe})$$

The preference φ_1 specifies that at some point, the agent will come close to a cliff, and immediately step away. Similarly φ_2 expresses that whenever the agent is unsafe, it will be safe at the next step. The solutions found can be seen on Figure 3c, where our algorithm returns three policies: the reference one (in blue), a policy satisfying φ_1 but not φ_2 (in yellow), and a policy satisfying both φ_1 and φ_2 (in red). Since this latter policy maximizes φ_1 and also satisfies φ_2, we have one fewer policy, compared to the two-preferences setting in Experiment 2. Both the yellow and red paths represent the shortest policies satisfying the properties.

Convergence Rates For experiments 1, 2 and 4, Figure 4 shows that our algorithm converges faster than the Q-learning algorithm towards optimal policy, meaning that it is more interesting to learn from an existing policy than to retrain an agent from scratch. Sometimes, the preferences expressed by the user may be conflicting, like in experiment 3 where the treasure is unsafe. Figure 5 shows that our algorithm converges easily, but this is not the case

for the Q-learning algorithm when rewards for both preferences are given (QL-ST). Specifying only one preference, either treasure (QL-T) or safe (QL-S), to the Q-learning algorithm speeds up the convergence, but our algorithm converges within as many steps; moreover, it even returns all three optimal policies within the same run (see Figure 1), removing the need for tuning the reward function. Indeed, QL-ST takes longer to converge since it oscillates between the two conflicting objectives, before settling for one.

To adapt the reward function passed to the Q-learning algorithm, so that we can learn similar behaviors as the ones that we expect with the temporal preferences expressed by the user, we sometimes had to modify the environment. For instance, to express the fact that the agent (previously) stepped on a `treasure` cell, we added a boolean tracking this in the environment. This was required because of a difference in expressivity of the reward languages of Q-learning and our setting, but also makes the task harder for our baseline, which can only learn stationary policies (and thus rely on the state as their only memory).

5.2 Restaurant

Episode During an episode, the agent navigates on a rectangular grid (cf. Figure 6), starting from the top-left corner with the aim to serve two tables, then throwing leftovers in the trash. The agent's state is defined as a tuple $s = (p, f, d, \tau_L, \tau_R)$ where p is the server position, f and d are respectively a boolean food-carrying flag and dish-carrying flag, τ_L and τ_R are respectively the left and right table states. A table state can take values in $\{waiting, served, dirty, clear\}$.

At each step, the agent selects one action from one of the cardinal directions, and `PICKUP`, `DELIVER`, `CLEAR`. Motion actions are deterministic and blocked by the walls. Action `PICKUP` is enabled only at `K`, the kitchen in the top-right corner, when food is not already carried and at least one table remains waiting. Action `DELIVER` is enabled only when the server is at a waiting table while carrying food, and changes that table from *waiting* to *served*. Action `CLEAR` has two context-dependent effects: at a *dirty* table it picks up dishes and sets the table to *clear*; at the trash, if dishes are carried, it disposes them. Served tables deterministically become dirty after a fixed delay (5 time steps), inducing a temporal coupling between servicing orders and cleaning.

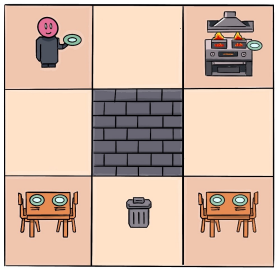


Figure 6: The layout of the restaurant. Tables are located at bottom-left and bottom-right corners, the server at top-left corner, the kitchen `K` at top-right corner and the trash in the bottom-mid cell of the map.

Reference Policy In the environment, the agent's objective is to serve both tables and clean up when they become dirty. Once all tables are in the *clear* state, the episode ends. The reference policy that we consider navigates to the kitchen to pick up food, serves the table on the right τ_R, returns to the kitchen to pick up another food item, serves the table on the left τ_L by going left from the kitchen, waits until the two tables are dirty, and finally clears dirty tables, starting with τ_R, by throwing leftovers in the trash. This takes a total of 27 steps. Obviously, this policy cannot be considered *efficient* due to the waiting time, and the trajectory could be optimized to take fewer steps.

Scenario 1: No Two Dirty Tables A first scenario is performed where the user preference is $\Box\neg(\langle\tau_L = dirty\rangle \wedge \langle\tau_R = dirty\rangle)$, meaning that the restaurant never has both tables dirty at the same time. The algorithm outputs two policies: the reference policy where two tables are dirty at the same time and a policy where τ_R is cleaned before τ_L is served, thus respecting the user preference. Note that the policy found is able to perform better than the reference policy, by ending the episodes two steps earlier. The convergence is achieved after 5×10^3 steps, taking about 10 seconds.

Scenario 2: Left Table First For the second scenario, one user preference is expressed via $\Diamond(\langle\tau_L = served\rangle \wedge \langle\tau_R = waiting\rangle)$, meaning that τ_L is prioritised by the server. As with the first scenario, the algorithm outputs two policies: the reference policy and a policy where τ_L is served first. The convergence is achieved after 21×10^3 steps, taking about 7 seconds.

In this section, we described a problem with more complex temporal dynamics. Because of this, we could not train through Q-learning, in reasonable time, an agent that complies with the user's preferences expressed in this section.

6 Discussion and Future Work

Preference Language Even though we only allow safe or co-safe LTL$_f$ formulas, the language can still express a wide range of desirable properties. Some non-decomposable LTL$_f$ formulas can be handled approximately within our framework. For example, a formula of the form $\Box(x \rightarrow \Diamond y)$ can be approximated by a formula $\Box(x \rightarrow (y \vee \bigcirc y \vee \cdots \vee \bigcirc^n y))$, which enforces eventual satisfaction within a bounded horizon.

A limit of the language concerns the intended semantics of temporal operators, especially $\Box$ and $\Diamond$. For instance, users may wish to interpret $\Box\,\varphi$ in different ways: either as a soft requirement that should be respected as often as possible, or as a strict requirement for which a single violation already constitutes failure, and subsequent violations of the properties are allowed. Distinguishing between these interpretations is important for aligning the formal specification with user intent, and making the policies returned as close

as possible to what the user envisioned. In future work, we wish to introduce this distinction directly in the language, and tailor the language to Problem 1, so that the user can best specify the intended behavior of the agent.

Our language is also expressive enough to capture explanation preferences often addressed by specialized methods. In particular, questions such as *"why was this action not taken in this state?"*, which call for contrastive explanations (Gajcin and Dusparic 2024), can be encoded directly in our language. This can be done by introducing suitable propositions at the relevant states.

More generally, our approach is compatible with alternative specification languages (Camacho et al. 2019), provided that they can be compiled into reward machines. One may even specify the reward machine directly, although this is significantly less intuitive than languages such as LTL_f.

It should also be noted that methods exists to synthesize LTL formulas from natural language (Fuggitti and Chakraborti 2023), whose use renders our method even more approachable to non-specialists.

Relationship with the Base Policy Our method requires a full input policy and does not work when only provided with a single plan (i.e. a single sequence of actions). The reason is that, once the agent deviates from the nominal trajectory to satisfy the user request, it must still be able to recover coherent behavior from newly reached states, and fall back to its initial intent. A plan that is defined only along one trajectory does not provide this recovery mechanism, whereas a policy does.

In the same vein, note that we do not know the original agent's intent. More formally, in this setting, we do not assume access to the original reward function. Instead, we rely on the observed behavior encoded by the base policy, while seeking policies that satisfy the temporal request and remain close to the reference behavior. If the base policy is well-defined on most reachable states, and effectively encodes a robust strategy for accomplishing the original task, then the agent can typically still achieve the underlying task, even after temporary deviations induced by the user preference.

Modelling the Base Policy In the current framework, the base policy is encoded through a dense reward signal, which provides frequent feedback whenever the agent's action aligns with the base policy. However, such a reward remains a proxy: it captures local action agreement rather than the deeper, long-term structure of the objective that originally shaped the policy. An interesting direction would therefore be to leverage Inverse Reinforcement Learning (IRL) to infer a more expressive reward representation from the observed behavior of π. Instead of simply rewarding step-wise agreement, IRL could recover a structured objective that encodes the policy at a trajectory level, revealing long-term outcomes. While this approach would significantly increase computational cost and introduce additional modeling complexity, it could yield a more faithful representation of the agent's intent.

7 Related Work

LTL for Task Specification The use of LTL formulas for task specification in stochastic (MDP) control and reinforcement learning is a topic that has attracted significant interest (e.g. (Ding et al. 2011; Wolff, Topcu, and Murray 2012; Fu and Topcu 2014; Camacho, Bienvenu, and McIlraith 2019; Aksaray et al. 2016; Li, Vasile, and Belta 2017; Camacho et al. 2019)). The general idea behind most of this work is to transform an LTL formula into an automaton, define an MDP that incorporates this automaton, and then obtain a policy that best satisfies the underlying LTL formula.

Early work in stochastic control (Ding et al. 2011; Wolff, Topcu, and Murray 2012; Fu and Topcu 2014) considers the LTL formula as a hard constraint, models this formula as a deterministic Rabin automaton, and focuses on the probabilistic guarantees of satisfaction. Relatedly, (Camacho, Bienvenu, and McIlraith 2019) proposes combining AI planning with LTL synthesis to exploit extended goals expressed in LTL, compiled into an automaton structure.

More recent work in RL (Aksaray et al. 2016; Li, Vasile, and Belta 2017; Camacho et al. 2019) uses an LTL formula as an explicit reward function. (Aksaray et al. 2016) propose a variant of Q-learning that allows a Signal Temporal Logic formula to be used as a reward function. In the same vein, (Li, Vasile, and Belta 2017) specify the task through a TLTL formula, a more expressive variant of SLTL, and use the robustness degree as a reward. (Camacho et al. 2019) propose translating an LTL (or other temporal logic) formula into a structured reward function, a reward machine.

Shielding A complementary line of work uses temporal logic specifications not as rewards, but as hard constraints on the agent's behavior. *Shielding* (Alshiekh et al. 2018) synthesizes, from a safety specification (typically expressed in LTL), a reactive component called the *shield*, that monitors the agent at each step and overrides any action that would lead to a violation of the specification. The shield guarantees safety throughout training and/or deployment without otherwise interfering with the underlying RL algorithm. While shielding and our approach both leverage temporal logic to shape RL agents, their end goals are orthogonal: shielding enforces hard safety constraints on a single policy, whereas we generate a diverse set of counterfactual policies satisfying user-provided LTL_f preferences to varying degrees, as these are viewed as soft constraints. Moreover, in addition to safety constraints, we also allow co-safety properties to be used as preferences.

Counterfactual Explanations With the taxonomy proposed in (Saulières 2025), our framework fits into the explanation of policies, more specifically into the group of works focused on policy comparison (Amitai and Amir 2022; Gajcin et al. 2021; Amitai, Septon, and Amir 2024). These different approaches propose using two policies and collecting trajectories starting from states where the action differs between the policies. The k most important trajectories are then proposed to the user (Amitai and Amir 2022; Amitai,

Septon, and Amir 2024) or the ones that are important up to a specific threshold (Gajcin et al. 2021). This importance is evaluated by a metric requiring access to Q-values. The metric can reflect importance in terms of the difference in performance of the agents (Amitai and Amir 2022; Amitai, Septon, and Amir 2024) or in terms of the difference in strategies adopted (Gajcin et al. 2021).

(van der Waa et al. 2018) fits into the explanation of sequences by comparing a counterfactual sequence inspired by the user's question in terms of outcomes. To generate the counterfactual sequence, a policy is obtained by locally modifying the agent's Q-value, taking into account the user's question. The user's input takes the form of one or more actions to be modified, whereas our approach proposes modeling a strategy using a rich language.

Our approach complements this group of studies by generating multiple policies for comparison. Furthermore, as the main focus is on policy generation, the comparison is carried out in a simple manner based on the general idea behind this group's work. We leave the potential integration of these methods into our framework to future work.

RL as an Explanation Framework Some works consider RL as a framework for providing explanations (Chen et al. 2022; Samoilescu, Looveren, and Klaise 2021; Nguyen et al. 2022; Lash 2024; Wang et al. 2018; Saulières, Cooper, and de Saint-Cyr 2023). The underlying principle of this framework is to model (at least in part) the problem of finding explanations as an RL problem. This approach has been used to generate counterfactual explanations of classification or regression models (Chen et al. 2022; Samoilescu, Looveren, and Klaise 2021; Nguyen et al. 2022; Lash 2024), explanations of recommendation systems (Wang et al. 2018), and even RL agents (Saulières, Cooper, and de Saint-Cyr 2023). To illustrate, (Lash 2024) relies on user preferences for features to obtain, via RL, an instance positioned on the decision boundary, and (Saulières, Cooper, and de Saint-Cyr 2023) uses RL to approximate the worst and best trajectories that an agent can take when performing a certain action from a state in a stochastic environment.

8 Conclusion

We introduced a framework for generating counterfactual policies guided by temporal logic preferences in reinforcement learning. Given a reference policy π, and without assuming any information about its training process, our approach enables users to explore counterfactual behaviors of π by expressing preferences as LTL_f formulas. By formulating the problem as a multi-objective task, i.e. balancing fidelity to the original policy and satisfaction of user preferences, we produce a diverse and rich set of policies.

To do so, we encode both the reference policy and temporal preferences as reward machines, then leverage a variant of PQL tailored to reward machines to find multiple policies on the Pareto front in a single training phase.

Future work includes integrating XRL methods for richer policy comparison, and proposing a specific language for *CFP-TL* to enhance the specification of user preferences.

AI Declaration

A Large Language Model was used to refine the writing of the paper, and to assist with the development of the implementation. All research ideas, theoretical foundations, experimental design, analysis, and core algorithmic contributions are original work by the authors.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Contribution Statement

All authors contributed equally.

References

Aksaray, D.; Jones, A.; Kong, Z.; Schwager, M.; and Belta, C. 2016. Q-learning for robust satisfaction of signal temporal logic specifications. In *55th IEEE Conference on Decision and Control, CDC 2016, Las Vegas, NV, USA, December 12-14, 2016*, 6565–6570. IEEE.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe reinforcement learning via shielding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2669–2678. AAAI Press.

Amitai, Y., and Amir, O. 2022. "I don't think so": Summarizing policy disagreements for agent comparison. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, 5269–5276. AAAI Press.

Amitai, Y.; Septon, Y.; and Amir, O. 2024. Explaining reinforcement learning agents through counterfactual action outcomes. In Wooldridge, M. J.; Dy, J. G.; and Natarajan, S., eds., *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, 10003–10011. AAAI Press.

Aronis, P. 2025. Leveraging reward machines for efficient multi-objective reinforcement learning. Master's thesis.

Bader, J., and Zitzler, E. 2011. Hype: An algorithm for fast hypervolume-based many-objective optimization. *Evolutionary computation* 19(1):45–76.

Camacho, A.; Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2019. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In Kraus, S., ed., *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 6065–6073. ijcai.org.

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2019. Towards a unified view of AI planning and reactive synthesis. In Benton, J.; Lipovetzky, N.; Onaindia, E.; Smith, D. E.; and Srivastava, S., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019*, 58–67. AAAI Press.

Chen, Z.; Silvestri, F.; Wang, J.; Zhu, H.; Ahn, H.; and Tolomei, G. 2022. Relax: Reinforcement learning agent explainer for arbitrary predictive models. In Hasan, M. A., and Xiong, L., eds., *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, 252–261. ACM.

Ding, X. C.; Smith, S. L.; Belta, C.; and Rus, D. 2011. LTL control in uncertain environments with probabilistic satisfaction guarantees. *CoRR* abs/1104.1159.

Fu, J., and Topcu, U. 2014. Probably approximately correct mdp learning and control with temporal logic constraints. *ArXiv* abs/1404.7073.

Fuggitti, F., and Chakraborti, T. 2023. NL2LTL – a Python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas. In *Thirty-Seventh AAAI Conference on Artificial Intelligence*, 16428–16430. AAAI Press.

Gajcin, J., and Dusparic, I. 2024. Redefining counterfactual explanations for reinforcement learning: Overview, challenges and opportunities. *ACM Comput. Surv.* 56(9):219:1–219:33.

Gajcin, J.; Nair, R.; Pedapati, T.; Marinescu, R.; Daly, E.; and Dusparic, I. 2021. Contrastive explanations for comparing preferences of reinforcement learning agents. *CoRR* abs/2112.09462.

Giacomo, G. D., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In Rossi, F., ed., *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 854–860. IJCAI/AAAI.

Hayes, C. F.; Radulescu, R.; Bargiacchi, E.; Källström, J.; Macfarlane, M.; Reymond, M.; Verstraeten, T.; Zintgraf, L. M.; Dazeley, R.; Heintz, F.; Howley, E.; Irissappane, A. A.; Mannion, P.; Nowé, A.; de Oliveira Ramos, G.; Restelli, M.; Vamplew, P.; and Roijers, D. M. 2022. A practical guide to multi-objective reinforcement learning and planning. *Auton. Agents Multi Agent Syst.* 36(1):26.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2001. Introduction to automata theory, languages, and computation. *Acm Sigact News* 32(1):60–65.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In Dy, J. G., and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 2112–2121. PMLR.

Lash, M. T. 2024. HEX: human-in-the-loop explainability via deep reinforcement learning. *Decis. Support Syst.* 187:114304.

Latvala, T. 2003. Efficient model checking of safety properties. In Ball, T., and Rajamani, S. K., eds., *Model Checking Software, 10th International SPIN Workshop. Portland, OR, USA, May 9-10, 2003, Proceedings*, volume 2648 of *Lecture Notes in Computer Science*, 74–88. Springer.

Lequen, A.; Legrand-Lixon, C.; and Saulières, L. 2026a. Pareto-Q-Learning with Reward Machines. *arXiv preprint*.

Lequen, A.; Legrand-Lixon, C.; and Saulières, L. 2026b. Supplementary material for the paper: "generating explainable counterfactual policies through temporal logic queries". DOI: 10.5281/zenodo.20010554.

Li, X.; Vasile, C. I.; and Belta, C. 2017. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, 3834–3839. IEEE.

Moffaert, K. V., and Nowé, A. 2014. Multi-objective reinforcement learning using sets of pareto dominating policies. *J. Mach. Learn. Res.* 15(1):3483–3512.

Nguyen, T. M.; Quinn, T. P.; Nguyen, T.; and Tran, T. 2022. Explaining black box drug target prediction through model agnostic counterfactual samples. *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 20(2):1020–1029.

Roijers, D. M.; Vamplew, P.; Whiteson, S.; and Dazeley, R. 2013. A survey of multi-objective sequential decision-making. *J. Artif. Intell. Res.* 48:67–113.

Samoilescu, R.; Looveren, A. V.; and Klaise, J. 2021. Model-agnostic and scalable counterfactual explanations via reinforcement learning. *CoRR* abs/2106.02597.

Saulières, L.; Cooper, M. C.; and de Saint-Cyr, F. D. 2023. Predictive explanations for and by reinforcement learning. In Rocha, A. P.; Steels, L.; and van den Herik, H. J., eds., *Agents and Artificial Intelligence - 15th International Conference, ICAART 2023, Lisbon, Portugal, February 22-24, 2023, Revised Selected Papers*, volume 14546 of *Lecture Notes in Computer Science*, 115–140. Springer.

Saulières, L. 2025. A survey of explainable reinforcement learning: Targets, methods and needs. *CoRR* abs/2507.12599.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement learning - an introduction, 2nd Edition*. MIT Press.

Towers, M.; Kwiatkowski, A.; Terry, J. K.; Balis, J. U.; Cola, G. D.; Deleu, T.; Goulão, M.; Kallinteris, A.; Krimmel, M.; KG, A.; Perez-Vicente, R.; Pierré, A.; Schulhoff, S.; Tai, J. J.; Tan, H.; and Younis, O. G. 2024. Gymnasium: A standard interface for reinforcement learning environments. *CoRR* abs/2407.17032.

van der Waa, J.; van Diggelen, J.; van den Bosch, K.; and Neerincx, M. A. 2018. Contrastive explanations for reinforcement learning in terms of expected consequences. *CoRR* abs/1807.08706.

Wang, X.; Chen, Y.; Yang, J.; Wu, L.; Wu, Z.; and Xie, X. 2018. A reinforcement learning framework for explainable recommendation. In *IEEE International Conference*

on Data Mining, ICDM 2018, Singapore, November 17-20, 2018, 587–596. IEEE Computer Society.

Watkins, C. J. C. H., and Dayan, P. 1992. Technical note Q-learning. *Mach. Learn.* 8:279–292.

Wiering, M. A., and de Jong, E. D. 2007. Computing optimal stationary policies for multi-objective markov decision processes. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 158–165.

Wolff, E. M.; Topcu, U.; and Murray, R. M. 2012. Robust control of uncertain markov decision processes with temporal logic specifications. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, 3372–3379.

Zhu, S., and Favorito, M. 2025. Lydiasyft: A compositional symbolic synthesis framework for ltlf specifications. In *Tools and Algorithms for the Construction and Analysis of Systems: 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3–8, 2025, Proceedings, Part I*, 295–302. Berlin, Heidelberg: Springer-Verlag.