Q-learning with Coupled Reward Machines for Unordered Tasks

Kristina Levina[1,2] **Nikolaos Pappas**[1] **Athanasios Karapantelakis**[2]
Aneta Vulgarakis Feljan[2] **Jendrik Seipp**[1]
[1]Linköping University, Linköping, Sweden [2]Ericsson Research
{kristina.levina, nikolaos.pappas, jendrik.seipp}@liu.se
{aneta.vulgarakis, athanasios.karapantelakis}@ericsson.com

Abstract

Reward machines (RMs) inform reinforcement learning agents about the reward structure of the environment, enabling support for non-Markovian tasks and improving sample efficiency. However, learning with RMs is ill-suited for long-horizon tasks where subtasks can be completed in any order because learnt information grows exponentially with the number of such subtasks. To address this, we introduce *Q-learning with coupled RMs* (QCoRM), a task-decomposition algorithm that learns one low-level policy per subtask and a high-level policy over their completion order. In QCoRM, learnt subtask-level information increases only linearly with the number of unordered subtasks, and we theoretically show that the global optimality guarantee of tabular Q-learning is preserved. To enable QCoRM, we introduce *agenda* and *coupled* RMs: agenda RMs identify mergeable task-progression states, and coupled RMs enable subtask-level policy sharing. We also introduce numeric RMs for compact task representation. Our experiments across four domains—featuring both discrete and continuous action and state spaces—demonstrate that QCoRM scales to more unordered subtasks than baseline algorithms.

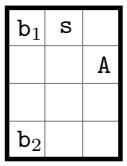
1 Introduction

Reinforcement learning (RL) agents learn policies—mappings from states to actions—by interacting with an environment and receiving reward-based feedback [34]. Designing effective reward functions to guide learning is often difficult and time-consuming [10]. Moreover, learning complex, long-horizon tasks is challenging because rewards are often sparse and delayed [9]. Task-decomposition methods mitigate these challenges by breaking down complex tasks into simpler subtasks [7]. One class of task-decomposition methods leverages logical formulae, referred to as *specifications*, to express the desired behaviour of the agent [18]. There are various specification languages and methods for compiling specifications into rewards [5].

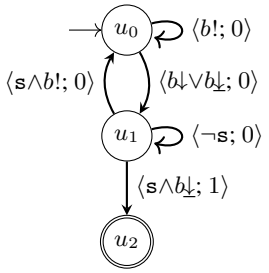
Another approach is to specify tasks in abstract graphs. Icarte et al. [16] formalised Boolean reward machines (RMs), automata in which states encode task progression. Augmenting environment states with RM states enables support for non-Markovian tasks. RMs have been shown to outperform plain reward functions and specifications in many settings by exploiting the task structure [29, 37]. Although existing RM-aware algorithms improve sample efficiency, complex, long-horizon tasks remain challenging, especially when subtasks can be completed in any order. In such cases, the number of RM states grows at least exponentially with the number of subtasks: an RM that tracks completion orders keeps one branch per order, and even one state per subset of completed subtasks yields 2^N states for N subtasks. Consequently, the agent must explore up to $N!$ distinct completion orderings to discover an optimal one. The resulting exponential state space makes learning difficult, and modelling an RM with so many states becomes tedious and error-prone.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

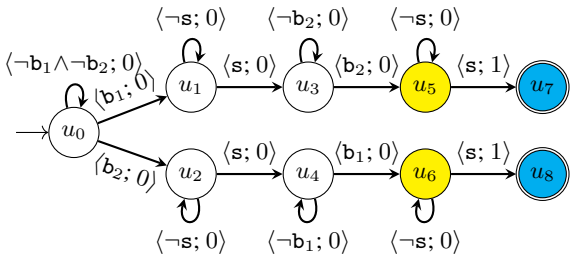
To address the exponential learning problem, we propose a new task-decomposition algorithm, *Q-learning with coupled RMs* (QCoRM), and two new types of RMs that enable it: *agenda* and *coupled* RMs. We illustrate them with a running example based on a variant of the Delivery domain—a deterministic finite-grid world with four-connected cells, some containing boxes. The agent must collect all boxes and deliver them to a designated station cell. Entering a box cell automatically collects the box, and the agent cannot accidentally drop it, but only one box can be carried at a time. Fig. 1a illustrates an example environment with two boxes. Fig. 1c depicts a Boolean RM for the introduced example, where we can see that, with an increasing number of boxes, the number of Boolean-RM states grows exponentially, and the agent must learn exponential information.



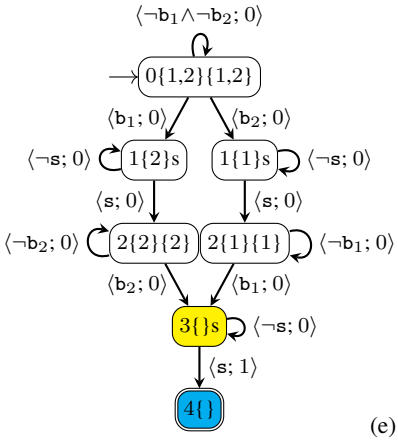
(a) Example Delivery instance with agent A, station s, and two boxes $\mathtt{b}_1$ and $\mathtt{b}_2$.



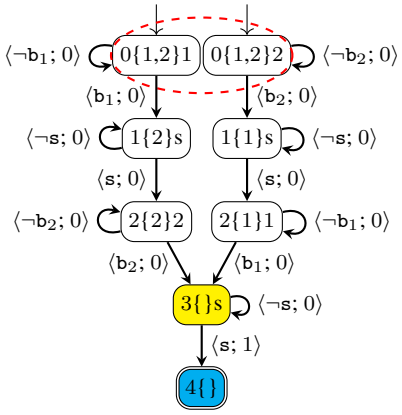
(b) Numeric RM.



(c) Boolean RM.



(d) Agenda RM.



State	η
0{1,2}1	12
0{1,2}2	10
1{2}s	9
1{1}s	6
2{2}2	8
2{1}1	2
3{}s	1
4{}	0

(e) Coupled RM. The red dashed line indicates that the agent is in states 0{1,2}1 and 0{1,2}2 concurrently. The η values were found using Q-learning with the coupled RM (QCoRM).

Figure 1: Delivery task (a) and its representations with reward machines (RMs) (b–e). In the numeric RM, a discrete numeric variable counts the number of uncollected boxes and is mapped to numeric feature b. Here, $b\downarrow$ is true when a box is collected in this step and uncollected boxes remain, $b\underline{\downarrow}$ is true when all boxes are collected, and $b!$ is true when no box is collected in this step and uncollected boxes remain. Boolean feature $\mathtt{s}$ is true when the agent arrives at the station. The agent receives a reward of 1 upon delivering all boxes to the station and 0 otherwise. In the Boolean, agenda, and coupled RMs, features $\mathtt{b}_1$ and $\mathtt{b}_2$ become true when the agent collects boxes 1 and 2, respectively. Symmetric states are shown in yellow and blue.

Not only learning exponential information but also manually specifying Boolean RMs with a possibly exponential number of states is problematic. We use numeric features in RMs to address this, and call the resulting RMs numeric RMs. In particular, unordered subtasks are encapsulated into a discrete numeric variable, which is then mapped to a numeric feature. For our running example, Fig. 1b shows a numeric RM for delivering two boxes, where numeric variable w with domain $\mathcal{D}_w = \{0, 1, 2\}$ counts the number of uncollected boxes. However, the agent cannot directly use this RM for learning because RM states u_0 and u_1 are visited twice, once for each collected box. Hence, the given numeric RM is translated to a Boolean RM (Fig. 1c).

Building on Boolean RMs, we label each RM state u with $\langle d, T, x\rangle$, where d is the distance from the initial state in the RM to u; T is an *agenda*, a set of remaining subtasks to be completed in any order; and x is the current objective. For conciseness, we abbreviate $\langle d, \{\tau_1, \tau_2, \dots\}, x\rangle$ as $d\{\tau_1, \tau_2, \dots\}x$.

States with identical labels are symmetric—they share a single Q-function—and can be merged. For example, in the agenda RM in Fig. 1d, yellow state $3\{\}$s corresponds to states u_5 and u_6 in the Boolean RM. State reduction accelerates learning by decreasing the number of distinct paths the agent must explore in the RM. Moreover, associating states with an agenda T allows us to split states by the subtasks in T, yielding a *coupled* RM (Fig. 1e). For example, if $T = \{\tau_1, \tau_2\}$, the state with current objective $x = T$ can be split into two coupled states: $d\{\tau_1,\tau_2\}\tau_1$ and $d\{\tau_1,\tau_2\}\tau_2$. Conceptually, the agent occupies all coupled states concurrently and may transition from any of them.

Once we obtain coupled RMs, we reduce the exponential learning bottleneck by exploiting the coupled-RM structure using a new task-decomposition Q-learning-based algorithm: QCoRM. In QCoRM, the agent learns a set of low-level policies for completing individual subtasks along with a high-level policy over coupled RM states to decide the completion order. The algorithm leverages the fact that each state in a coupled RM corresponds to exactly one subtask. For the high-level policy, we treat coupled RM states as arms in a multi-armed bandit. As the decision metric, we use the expected number of steps $\eta(u)$ from each RM state u to a goal state. The high-level policy coordinates low-level learning by shaping the reward signals using episode length, preserving global optimality in tabular settings. In our running example, the QCoRM agent learns low-level policies to locate box 1, box 2, and the station, as indicated by the right parts of the RM-state labels: 1, 2, and s, respectively. In coupled states $0\{1,2\}1$ and $0\{1,2\}2$, the agent decides to transition from $0\{1,2\}2$, i.e. collect box 2 first, because its η value is 10, which is lower than 12 for state $0\{1,2\}1$.

While we focus on unordered subtasks, QCoRM can also reduce the number of low-level policies to learn in sequential tasks. For example, to reach `a`, `b`, and `a` in order (RM states u_0, u_1, and u_2), the QCoRM agent learns two reusable policies to reach `a` and `b`. In contrast, the Boolean-RM agent learns three policies, one per RM state u_0, u_1, and u_2.

2 Background

We begin by providing background information on RL and RMs. For more details on these topics, we refer the reader to Sutton and Barto [34] and Icarte et al. [16], respectively.

Single-agent RL tasks are formalised via Markov decision processes (MDPs), defined by the tuple $M = \langle S, s_0, A, p, r, \gamma \rangle$, where S is a finite set of environment states, $s_0 \in S$ is an initial state, A is a finite set of actions, $p : S \times A \to \Delta(S)$ is a transition probability function, $\Delta(S)$ is the probability simplex over S, $r : S \times A \times S \to \mathbb{R}$ is a reward function, and $\gamma \in (0, 1)$ is a discount factor. In state s_t, the agent performs action a_t according to policy $\pi(a_t|s_t)$, transitions to state s_{t+1} with probability $p(s_{t+1}|s_t, a_t)$, and receives reward r_{t+1}. The process repeats until episode termination. The objective is to find an optimal policy $\pi^*(a_t|s_t = s)$ for all $s \in S$ to maximise the expected return $\mathbb{E}_\pi[\sum_{k=0}^{K-1} \gamma^k r_{t+k+1}|s_t = s]$, where K is the episode length.

The Q-function $q^\pi(s, a)$ quantifies the expected return the RL agent can achieve by taking a specific action a in a given state s and following a policy π thereafter. Formally,

$$q^\pi(s, a) = \mathbb{E}_\pi[\sum_{k=0}^{K-1} \gamma^k r_{t+k+1}|s_t = s, a_t = a]. \tag{1}$$

For an optimal policy π^*, $q^* = q^{\pi^*}$. Tabular Q-learning is an off-policy algorithm that estimates $q^*(s, a)$ without knowledge of the transition function p [40]. The Q-values are learnt through environment interaction and are guaranteed to converge to $q^*(s, a)$ under infinite exploration of all state–action pairs. The algorithm initialises a Q-table randomly for all (s, a). The $Q(s, a)$ values are then updated at each iteration i:

$$Q_{i+1}(s, a) \overset{\alpha}{\leftarrow} r(s, a, s') + \gamma \max_{a'} Q_i(s', a'), \tag{2}$$

where α is a learning rate. The notation $x \overset{\alpha}{\leftarrow} y$ is expanded as $x \leftarrow x + \alpha(y - x)$.

Double deep Q-learning (DDQN) [38] extends Q-learning to large or continuous state spaces with discrete actions, and twin delayed deep deterministic policy gradient (TD3) [11] extends it to continuous action spaces, both at the cost of global optimality guarantees [26].

A **Boolean RM** is a finite-state automaton that encapsulates the reward structure of the environment. Transitions between RM states specify the rewards received by the agent.

Definition 1 (Boolean reward machine)**.** *A* Boolean *RM is a tuple* $\mathcal{R}^{Bool}_{\mathcal{P}SA} = \langle U, u_0, F, \delta_u, \delta_r \rangle$ *given sets of propositional symbols* $\mathcal{P}$*, environment states* S*, and actions* A*. In the tuple,* U *is a finite set of states,* u_0 *is an initial state,* F *is a finite set of terminal states,* δ_u *is a state-transition function such that* $\delta_u : U \times 2^{\mathcal{P}} \to U \cup F$*, and* δ_r *is a state-reward function such that* $\delta_r : U \to [S \times A \times S \to \mathbb{R}]$.

Following Icarte et al. [16], a labelling function $L : S \times A \times S \to 2^{\mathcal{P}}$ maps each environment experience (s, a, s') to the set of true propositions; δ_u then selects the abstract successor RM state; and δ_r assigns the reward. Intuitively, an MDP with RMs (MDPRM) is an MDP defined over the cross-product $\tilde{S} = S \times (U \cup F)$: an MDPRM is a tuple $M_{\mathcal{R}} = \langle \tilde{S}, \tilde{s}_0, \tilde{A}, \tilde{p}, \tilde{r}, \tilde{\gamma} \rangle$, where $\tilde{s}_0 \in \tilde{S}$ is an initial state; $\tilde{A} = A$; state-transition function $\tilde{p}(\langle s', u' \rangle | \langle s, u \rangle, a)$ is $p(s'|s, a)$ if $u' = \delta_u(u, L(s, a, s'))$ and $u \in U$, $p(s'|s, a)$ if $u' = u$ and $u \in F$, and 0 otherwise; state-reward function $\tilde{r}(\langle s, u \rangle, a, \langle s', u' \rangle)$ is $\delta_r(u)(s, a, s')$ if $u \notin F$ and 0 otherwise; and $\tilde{\gamma} = \gamma$ is a discount factor. The task formulation with respect to MDPRMs is Markovian. Thus, Q-learning (among other RL methods) can be used to solve it. Optimal-solution guarantees of RL algorithms for MDPRMs are the same as for regular MDPs.

Icarte et al. [16] formalised Q-learning algorithms for MDPRMs. One is Q-learning with RMs (QRM) over the cross-product $S \times U$. To exploit the RM reward structure, QRM with counterfactual reasoning (CRM) and a sub-optimal options-based hierarchical RL algorithm, HRM, have been developed. In CRM, synthetic experiences are generated for each RM state per environment interaction.

3 Numeric, Agenda, and Coupled Reward Machines

We now formally introduce numeric, agenda, and coupled RMs and compare them to Boolean RMs. Numeric RMs compactly model changes in numeric variables, simplifying the RM design process. For example, in the Delivery domain (Fig. 1b), the numeric variable counts uncollected boxes—remaining subtasks. More generally, numeric variables could encode any discrete numeric information, such as remaining resources. Numeric RMs cannot support continuous numeric variables, though, as the induced Boolean RM would have infinitely many states. Thus, we assume the following.

Assumption 1. *All numeric variables in numeric RMs are discrete and have finite bounds.*

We introduce numeric features inspired by qualitative numeric planning [33], where actions increase or decrease numeric variables by unspecified amounts. In our setting, numeric variables give a sense of task progression to the agent, so we make the following assumption.

Assumption 2. *Numeric variables in numeric RMs never increase.*

Each numeric variable $w \in W$ is lower-bounded by w_ℓ, which specifies its goal value. We map each w to a numeric feature p_w with domain $\mathcal{D}_{p_w} = \{\downarrow, \underline{\downarrow}, !\}$. Value $p_w\downarrow$ indicates that w has decreased since the previous observation but remains above w_ℓ. Value $p_w\underline{\downarrow}$ indicates that w has reached w_ℓ. Value $p_w!$ indicates that w has neither decreased nor reached w_ℓ.

Proposition 1. *Under Assumption 2, domain* $\mathcal{D}_{p_w} = \{\downarrow, \underline{\downarrow}, !\}$ *fully captures possible changes between the current and previous values of* w*, i.e.* w_t *and* w_{t-1} *(Proof in Appendix A).*

Now, we have all the ingredients to define numeric RMs.

Definition 2 (Numeric reward machine)**.** *A* numeric RM *is a tuple* $\mathcal{R}^{num}_{\mathcal{P}SA} = \langle U, u_0, F, \delta_u, \delta_r \rangle$ *given sets of Boolean and numeric features* $\mathcal{P} = \mathcal{P}_{Bool} \cup \mathcal{P}_{num}$*, environment states* S*, and actions* A*.* U*,* u_0*,* F*, and* δ_r *are defined as in a Boolean RM; and* $\delta_u : U \times 2^{\mathcal{P}_{Bool}} \times 3^{\mathcal{P}_{num}} \to U \cup F$ *is a state-transition function. Each numeric feature corresponds to a discrete numeric variable* $w \in W$ *with domain* $\mathcal{D}_w$.

Features encode observable environment events, and a numeric RM accepts both Boolean and numeric features. Boolean features are propositional symbols that the agent can evaluate at each state.

Assumption 3. *For any Boolean feature* p_B *and state* s*, the agent has access to* $[\![p_B]\!]$.

Here, $[\![]\!]$ is the Iverson bracket. This common assumption ensures that the agent's traversal through the RM aligns with actual environment conditions. If the agent does not have information about the true evaluation of some propositional symbol relevant for the task, the problem becomes partially observable and requires partially observable RMs [24, 39]. Likewise, we assume that the agent has access to the values of each numeric feature $p_w \in \mathcal{D}_{p_w}$. That is, the agent can observe whether w has decreased since the previous observation and whether the goal w_ℓ has been reached.

Numeric RMs offer a compact way to specify tasks. They are particularly useful for tasks in which subtasks can be completed in any order and the agent must learn an optimal ordering. However, an RL agent cannot learn a policy directly from a numeric RM because a single RM state may correspond to multiple stages of task progression. For instance, in the example numeric RM (Fig. 1b), the agent can occupy only two RM states before task completion, regardless of the number of boxes to deliver. Specifically, different trajectory segments—corresponding to different numbers of undelivered boxes—all map to the same numeric-RM state u_0. This would cause the agent to repeatedly overwrite learnt information for u_0. Therefore, numeric RMs are translated to Boolean RMs by unfolding their states with respect to the domains of numeric variables $\mathcal{D}_w$ under the assumptions below.

Assumption 4. *The domains of numeric variables in a numeric RM correspond to the collections of different subtasks given to the agent.*

To enable the functionality of subtask completion in any order, we specifically assume the following:

Assumption 5. *Sets of subtasks encapsulated in numeric variables can be completed in any order, and subtasks corresponding to Boolean features are ordered.*

Therefore, we assume that each subtask in the collection of ordered subtasks is assigned a Boolean feature; i.e. ordered parts of the task stay outside the numeric variable. Several groups of unordered subtasks for sequential completion correspond to distinct numeric features, and the RM is constructed accordingly. In contrast, completion of several distinct groups of unordered subtasks in parallel is not currently supported and would require tracking their task progression in parallel.

Assumption 6. *A numeric RM is formulated such that only one subtask for each feature p_w can be completed per step in the environment and, upon its completion, $p_w{\downarrow}$ or $p_w{\downarrow}$ signals to transition to the next RM state.*

If the agent were allowed to complete several objectives simultaneously, then it would need to fan out over the completed subset, which could be added to QCoRM in future work.

If the assumptions are met, a translation algorithm takes a numeric RM, $\mathcal{R}^{\text{num}}_{\mathcal{P}SA}$, along with the numeric-variable domains $\mathcal{D}_w$ as input and outputs a Boolean RM, $\mathcal{R}^{\text{Bool}}_{\mathcal{P}SA}$. The RM states are unfolded with respect to $\mathcal{D}_w$ following the given $\mathcal{R}^{\text{num}}_{\mathcal{P}SA}$ transitions. The unfolding tracks the completion order, so each $\mathcal{R}^{\text{Bool}}_{\mathcal{P}SA}$ state corresponds to a sequence of completed subtasks rather than to their set. Thus, $\mathcal{R}^{\text{Bool}}_{\mathcal{P}SA}$ keeps one branch per completion order and grows as $\Theta(N!)$ for N unordered subtasks. Moreover, $\mathcal{R}^{\text{Bool}}_{\mathcal{P}SA}$ can contain symmetric states corresponding to the same Q-values. Symmetric states could be merged by hand, but locating them systematically requires relating RM states to the additional information about uncompleted subtasks and completion orders.

Another limitation of Boolean RMs is the partial use of CRM in domains where completed subtasks are removed from the environment. Once a subtask is completed, the agent will simulate incorrect counterfactual experiences for RM states associated with that subtask. For example, in the toy task depicted in Fig. 1a, the agent transitions to u_1 in the Boolean RM (Fig. 1c) upon collecting box 1. The cell containing $\mathtt{b}_1$ is then cleared, and future visits to this cell will not make Boolean feature $\mathtt{b}_1$ true. Consequently, counterfactual experiences for states u_0 and u_4 will be erroneous. Therefore, counterfactual reasoning can only be applied to RM states reachable after u_1. We handle the problems above by associating the RM states with an *agenda*—the remaining subtasks to complete.

Definition 3 (Agenda reward machine)**.** *An agenda RM is a Boolean RM translated from a numeric RM with numeric variables $w \in W$, where each w encapsulates a set $\mathcal{T}_w$ of unordered subtasks, together with a state-annotation function $\Xi : U \cup F \to \{\langle d, (T_w)_{w\in W}, x\rangle \mid T_w \in \mathcal{P}(\mathcal{T}_w)\}$, in which states with identical annotations are merged, where $\mathcal{T}_w$ is the set of all unordered subtasks encapsulated in w, $T_w \subseteq \mathcal{T}_w$ is the set of the remaining subtasks of w, d is the RM state depth, and x is the current objective. The depth d of RM state u is the distance from the initial state of the RM to u.*

We construct the power set $\mathcal{P}(\mathcal{T}_w)$ for each $w \in W$ assuming that the agent needs to find an optimal completion order. In $\langle d, (T_w)_{w\in W}, x\rangle$, x can be either the agenda T_w of some $w \in W$ or a Boolean feature. If $x = T_w$, then, upon completion of any subtask from T_w, the agent transitions to the next RM state. This annotation scheme aligns with the RL principle of learning from information contained in future states (see the Q-learning update in (2)). By tracking remaining subtasks rather than completed ones, the RM maintains the correct direction of information flow for value backpropagation. Furthermore, since the label forgets the order of the completed subtasks, assigning symmetric states identical labels requires tracking the RM state depth.

Definition 4 (Task progression)**.** *Let Σ be the set of subtask sequences that complete a task. When the agent has completed a prefix h of a sequence in Σ, its task progression is the triple $\langle X, |h|, \Sigma_h \rangle$, where X is the set of possible subtasks to pursue next in the task sequences, $|h|$ is the number of completed subtasks, and $\Sigma_h = \{\sigma \mid h\sigma \in \Sigma\}$ is the set of remaining subtask sequences.*

For example, states u_5 and u_6 in the Boolean RM (Fig. 1c) are reached after sequences $\langle \mathtt{b}_1, \mathtt{s}, \mathtt{b}_2 \rangle$ and $\langle \mathtt{b}_2, \mathtt{s}, \mathtt{b}_1 \rangle$, respectively, but correspond to the same task progression $\langle \{\mathtt{s}\}, 3, \{\langle \mathtt{s} \rangle\} \rangle$.

Proposition 2. *Under Assumptions 2 and 4–6, the annotation scheme in an agenda RM uniquely identifies RM states with respect to task progression (Proof in Appendix B).*

Agenda RMs also make CRM more effective in domains where completed subtasks are removed from the environment. Counterfactual experiences are still simulated only for RM states reachable from the current state, but each merged state stands for all completion orders leading to it. For example, after collecting box 1, the agent is in state $1\{2\}$s of the agenda RM (Fig. 1d), from which states $2\{2\}\{2\}$, $3\{\}$s, and $4\{\}$ are reachable. These three states correspond to states u_3, u_5, u_6, u_7, and u_8 in the Boolean RM, whereas only u_3, u_5, and u_7 are reachable from u_1.

We translate the given numeric RM into an agenda RM using a similar algorithm to that of numeric-to-Boolean-RM translation. The translation differs only in the annotation scheme. Merging states with the same labels removes the factorial factor, but agenda RMs still have $\Theta(2^N)$ states because their labels are indexed by subsets of remaining subtasks: in Delivery with N boxes, there are $2^{N+1} - 1$ agenda-RM states, e.g. 511 against 219 201 Boolean-RM states for $N = 8$. Converting the agenda RM to a coupled RM does not reduce the state count further but enables per-subtask policy sharing.

Definition 5 (Coupled reward machine)**.** *A coupled RM is an agenda RM where each state with current objective $x = T_w = \{\tau_1, \tau_2, \ldots, \tau_m\}$, $w \in W$, is split into $m = |T_w|$ states that have the same depth and agendas as the original state and current objectives $\tau_1, \tau_2, \ldots, \tau_m$, respectively. The split states remain coupled such that the agent is in all of them concurrently.*

Algorithmically, the agent occupies all coupled-RM states simultaneously while being in a single environment state. Moreover, because the agent has access to the RM label and each coupled-RM state corresponds to exactly one subtask, the agent can learn low-level policies for completing individual subtasks. Such subtask-policy sharing is not possible in Boolean or agenda RMs, where states can have several outgoing transitions, e.g. u_0 in Fig. 1c, and thus are not related to single subtasks.

Since groups of unordered subtasks are completed one after another, the agent pursues at most one agenda at a time, while the agendas of all other numeric variables remain unchanged. Therefore, without loss of generality, we consider a single numeric variable from now on and drop the subscript w, i.e. RM states are annotated with $\langle d, T, x \rangle$, where $T \subseteq \mathcal{T}$.

4 Q-learning with Coupled Reward Machines

We introduce Q-learning with coupled RMs (QCoRM), a new task-decomposition algorithm in which the agent learns low-level policies for completing individual subtasks and a high-level policy for determining their completion order.

The **high-level policy** selects the next subtask from the remaining set $T = \{\tau_1, \ldots, \tau_m\}$ by treating the available choices as arms in a stochastic multi-armed bandit [34]. Because costs change as low-level policies improve (non-stationarity), we maintain exponentially recency-weighted estimates via a constant step-size incremental update. Let $\eta(u)$ be the estimated number of steps from each RM state u to a goal. After each episode e, for every RM state $u_e \in U_e$ visited during that episode, we update $\eta(u_e) \xleftarrow{\alpha_\eta} K_{u_e \to \text{goal}}$, where α_η is a learning rate and $K_{u_e \to \text{goal}}$ is the observed number of steps from u_e to the goal. We store $\eta(u)$ in a lookup table with $|U|$ entries. When exploiting, from the current set of coupled states, the agent selects the transition with the lowest cost as the next subtask. Once the agent converges to an optimal solution, the stored $\eta(u)$ values approximate the optimal average steps-to-go, and the high-level policy yields a trajectory of average length $\bar{K}_{\text{op}}$. To force exploration of all transitions in the RM, we introduce an exploration probability $\xi \in (0, 1]$. During exploration, the agent randomly selects RM transitions with a bias towards unused transitions.

Each state in a coupled RM is associated with exactly one subtask, allowing the agent to learn a separate **low-level policy** for each subtask in $U' = \mathcal{T} \cup B$, where $\mathcal{T}$ is the complete agenda and B

is the set of all Boolean objectives. Consequently, the state space of the corresponding MDPRM can be constructed as $S \times U'$ using the set of all subtasks U' rather than the full set of RM states U. The amount of low-level information to learn thus increases only linearly with $|U'|$. In particular, the number of low-level policies, simulated experiences replayed per step, and, in the tabular case, the Q-table size all grow only linearly with $|U'|$, so the learning bottleneck scales linearly with the number of unordered subtasks. In contrast, Boolean RMs require the full state space $S \times U$, forcing the agent to learn information exponential in the number of unordered subtasks. Importantly, the high-level policy over the coupled RM indexes its cost estimates $\eta(u)$ by RM state $u \in U$, and $|U|$ still grows exponentially with the number of unordered subtasks. This does not affect learning efficiency, though, as each $\eta(u)$ is a single scalar updated in $O(1)$ with no environment interaction.

As the agent occupies all coupled states $\langle d, T, \tau_1 \rangle$, $\langle d, T, \tau_2 \rangle$, $\ldots$, $\langle d, T, \tau_m \rangle$ at once, it can perform Q-learning updates in parallel. As long as the agent remains in the current set of coupled RM states, the Q-values for each subtask τ_k, $k = 1, \ldots, m$, are updated concurrently:

$$Q_{i+1}(\langle s, \tau_k \rangle, a) \xleftarrow{\alpha} r(\langle s, \tau_k \rangle, a, \langle s', \tau_k \rangle) + \gamma \max_{a' \in A} Q_i(\langle s', \tau_k \rangle, a'), \tag{3}$$

where α is the learning rate, γ is the discount factor, and $r(\langle s, \tau_k \rangle, a, \langle s', \tau_k \rangle)$ is the immediate reward. When the agent completes subtask τ_ℓ and transitions to the next RM state, the Q-values associated with the remaining subtasks $T \setminus \{\tau_\ell\}$ are updated according to (3). The update for RM state $\langle d, T, \tau_\ell \rangle$ is, however,

$$Q_{i+1}(\langle s, \tau_\ell \rangle, a) \xleftarrow{\alpha} R(K, K^{\tau_\ell}), \tag{4}$$

where K is the episode length in timesteps, K^{τ_ℓ} is the τ_ℓ execution duration in timesteps, and $R(K, K^{\tau_\ell})$ is the final reward for τ_ℓ completion.

Theorem 3. *The final reward for completing any subtask τ preserves the global optimality guarantee of tabular Q-learning if it is defined as*

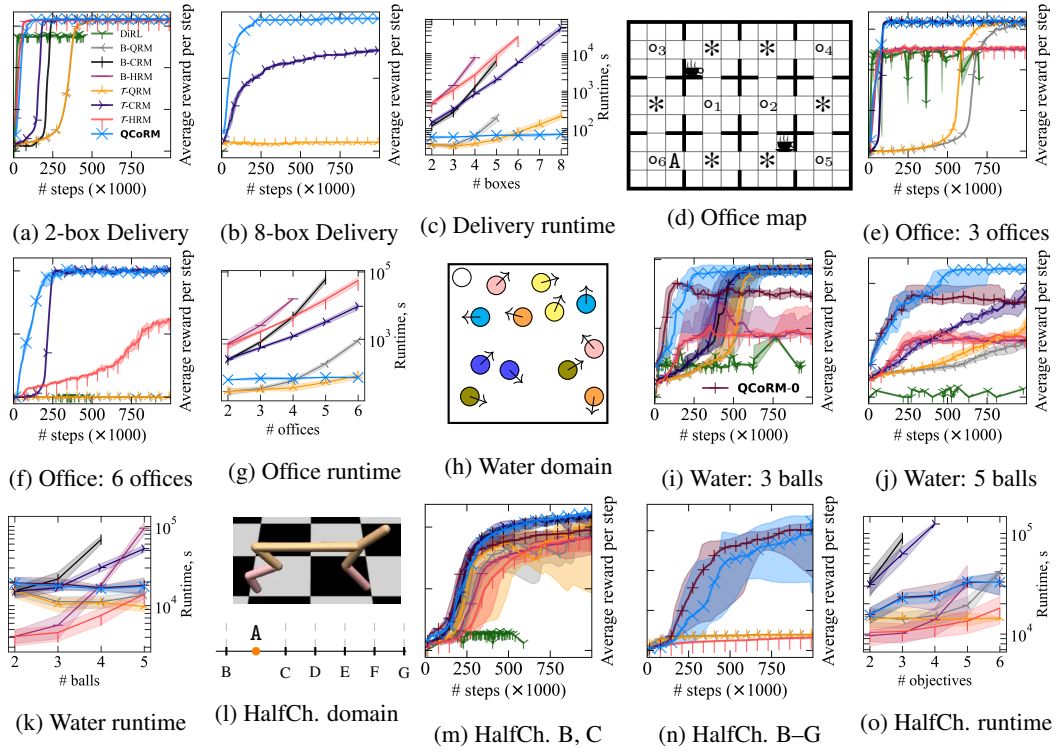
$$R(K, K^\tau) = \begin{cases} \check{R} & \text{if } \delta K^\tau = 0, \\ \gamma^{\delta K^\tau + 1} \check{R} + \frac{\gamma^{1-K^\tau} - \gamma^{\delta K^\tau}}{1-\gamma} r_{mi} & \text{otherwise}, \end{cases} \tag{5}$$

where K is the episode length, K^τ is the τ execution duration in timesteps, $\check{R} \in \mathbb{R}_{>0}$, $\delta K^\tau = \max\{0, \bar{K}^\tau_{op} - K^\tau, K - \bar{K}_{mi}\}$, $\bar{K}^\tau_{op}$ is the expected τ duration under a globally optimal policy (updated during the high-level exploitation as $\bar{K}^\tau_{op} \xleftarrow{\alpha_K} K^\tau$), $\bar{K}_{mi}$ is the minimum episode length (updated during the high-level exploitation as $\bar{K}_{mi} \xleftarrow{\alpha_K} K$ if $K < \bar{K}_{mi}$), α_K is the learning rate, and $r_{mi} \leq 0$ is the minimum immediate reward (Proof and a worked example in Appendix C).

Valuable experiences with transitions between distinct RM states (4) are stored in a temporary buffer during each episode. Once the episode terminates, i.e. the episode length and the low-level-policy durations are known, the rewards in the stored valuable experiences are reshaped with (5). In particular, the global episode-length term $K - \bar{K}_{\text{mi}}$ in δK^τ in the reward makes each local policy aware of the global context so that it is not greedily optimised for the local duration. When the agent exploits, i.e. when it selects the best transition based on the lowest $\eta(\langle d, T, \tau_k \rangle), k = 1, 2, \ldots, m$, the reshaped-reward experiences are added to the buffer and are replayed normally. In this way, the low-level policies are trained to align with the best high-level policy, converging to optimal episode lengths. Algorithm 1 in Appendix E summarises QCoRM.

5 Experiments

We integrate QCoRM with DDQN and TD3 by adding a dedicated replay buffer that stores valuable experiences corresponding to the transitions between different RM states. After each exploitation episode, these stored transitions are replayed to train the network using the same update procedure as above. Our implementation builds on the code by Icarte et al. [16] and Stable-Baselines3 [30]. All code and experiment data are accessible on Zenodo [23]. For evaluation, we use two tabular domains, Delivery (our running example) and Office [16], and two continuous domains, Water [21] and MuJoCo HalfCheetah [35]. The agent's initial position is fixed throughout the experiments. Experiments run on Intel Xeon Gold 6130 CPUs, using a single core with 16 GiB of memory for the tabular domains and 32 GiB for the continuous domains. Each algorithm run is limited to 10^6



(a) 2-box Delivery (b) 8-box Delivery (c) Delivery runtime (d) Office map (e) Office: 3 offices

(f) Office: 6 offices (g) Office runtime (h) Water domain (i) Water: 3 balls (j) Water: 5 balls

(k) Water runtime (l) HalfCh. domain (m) HalfCh. B, C (n) HalfCh. B–G (o) HalfCh. runtime

Figure 2: Results of QCoRM (ours) and the baselines: Q-learning (QRM), QRM with counterfactual reasoning (CRM), hierarchical RL (HRM), and DiRL. Boolean RMs (B-QRM, B-CRM, and B-HRM) and our agenda RMs ($\mathcal{T}$-QRM, $\mathcal{T}$-CRM, and $\mathcal{T}$-HRM) were tested on Delivery (a,b), Office (e,f), Water (i,j), and HalfCheetah (m,n). As DiRL does not allow setting the number of environment interactions in advance and provides a complete evaluation only after training finishes, we cannot plot DiRL's learning curve. Instead, we run DiRL with an increasing number of iterations and mark its final performance as a single point per run against the total environment interactions consumed—summing both low-level policy training and high-level policy construction. QCoRM-0 stands for QCoRM without the joint optimisation from Eq. (5). (c,g,k,o) Time (in seconds) to run 10^6 steps in relation to the number of unordered objectives. (d) Office environment with agent A, offices $\mathtt{o}_i, i = 1, 2, \ldots, 6$, decorations marked by stars, and coffee machines. (h) Water environment with the agent (white ball) and balls of different colours with arrows indicating their velocities. (l) HalfCheetah environment with agent A (orange dot) and objectives B, C, D, E, F, and G along the horizontal axis.

steps and 40 hours. We report median performance across 10 seeds, with the shaded regions on plots representing the 25th and 75th percentiles.

We compare QCoRM against state-of-the-art RM methods (CRM, HRM, and QRM) and DiRL [19], a compositional learning approach from specifications. We use the DiRL code from the authors' repository and translate agenda RMs into abstract reachability graphs required by DiRL. For a fair comparison, we use the same low-level policy learners for DiRL and QCoRM. For the tabular and Water domains, the agent receives a reward of 1 after task completion and 0 otherwise; we adopt this sparse reward setting because positive intermediate rewards can lead to sub-optimal policies [6]. For HalfCheetah, the agent receives a reward of 1 000 upon completing each subtask and a control penalty otherwise. The architecture and hyperparameters are described in Appendix D, and all experimental plots for all numbers of unordered subtasks are provided in Appendix F. Finally, we show experimental results for the Q-value-based, discounted-upper-confidence-bound (UCB), and Thompson sampling high-level controllers integrated in QCoRM in Appendix G for all four domains.

Our **Delivery** tasks are seven 10×10 grids with 2–8 randomly placed boxes that are removed upon collection. Figs. 2a and 2b show experimental results for two and eight boxes, respectively. Although HRM converges fastest for two boxes, it fails to converge for eight boxes. QCoRM converges to an optimal policy for two and eight boxes, followed by the CRM methods. Notably, QCoRM's

performance does not deteriorate after the removal of collected boxes. In contrast, the CRM methods simulate experiences only for reachable RM states, so fewer experiences are available as the agent traverses the RM. Convergence is faster for CRM with the agenda RMs than with the Boolean RMs because more states can be identified as reachable due to the informative annotation scheme. Boolean RMs for tasks with more than five boxes exceed available memory. DiRL's final performance remains sub-optimal for all tested numbers of iterations and all task sizes.

The **Office** domain [16] (Fig. 2d) is a grid world with walls, doors, and decorations. The agent is given five progressively more complex tasks: deliver coffee to 2–6 offices in any order. The agent fails the episode upon stumbling on a decoration. Figs. 2e and 2f show the results for delivering coffee to three and six offices, respectively. QRM-based methods converge slowest. DiRL's final performance remains sub-optimal. HRM converges to sub-optimal policies. As in Delivery, QCoRM outperforms CRM-based methods. Unlike in Delivery, CRM with both the agenda and Boolean RMs converges at similar rates because offices are not removed from the map after the agent's visit, keeping all the RM states reachable for experience simulation.

The **Water** domain [21] is a two-dimensional arena populated by coloured balls that move at fixed speeds and rebound off walls (Fig. 2h). The agent, represented by a white ball, adjusts its velocity along the four cardinal directions via discrete control. The domain is challenging because the order in which balls are touched affects the low-level policies for reaching each individual ball. For example, the agent may need to decelerate before touching a ball to optimally position itself for the next. As ball positions and velocities are continuous, we use DDQN. We randomly generate maps with six pairs of differently coloured balls and define four progressively more difficult tasks to touch two to five balls of specified colours in any order without touching others. Figs. 2i and 2j show the results for three and five balls, respectively. QCoRM converges fastest, followed by CRM, then HRM (sub-optimal) and QRM, whereas DiRL's final performance remains sub-optimal. To show that the joint optimisation with the reward function (5) in QCoRM ensures convergence to an optimal policy, we perform an **ablation** and give the agent a reward of 1 for completing a subtask in place of (5). The results in Figs. 2i and 2j confirm that the agent without the joint optimisation (QCoRM-0) converges to sub-optimal policies, as the low-level policies are not shaped by the global episode-length signal.

The **HalfCheetah** domain [35] involves controlling a two-dimensional bipedal robot to reach specific positions along a horizontal axis (Fig. 2l). We define progressively more difficult tasks: reaching two to six positions in any order. We use TD3 for continuous control. CRM-based methods, on par with QCoRM, learn to reach two positions quickest (Fig. 2m). Notably, QCoRM-0 without joint optimisation learns to reach the leftmost position B without decelerating, resulting in a sub-optimal solution. Reaching six positions (Fig. 2n) can only be done by QCoRM and QCoRM-0.

We compare the **runtimes** of the tested algorithms against task size in Figs. 2c, 2g, 2k, and 2o. In tabular domains, QRM runtimes scale exponentially due to increasing Q-table sizes. In continuous domains, QRM runtimes change little with agenda RMs but grow for B-QRM in HalfCheetah because the RM state is one-hot encoded in the network input. The plots show that QCoRM runtimes scale linearly because the number of simulated experiences grows linearly with the number of subtasks. In HalfCheetah, QCoRM runtimes increase stepwise because the TD3 batch and replay-buffer sizes are scaled by $\lceil N/2 \rceil$, the average number of coupled states per step. In contrast, both HRM and CRM runtimes exhibit exponential growth because the number of simulated experiences is proportional to the number of RM states. Notably, the exponential blowup in the number of Boolean RM states causes B-CRM and B-HRM to exceed the time or memory limits in large tasks. While the CRM- and HRM-based methods are therefore unsuitable for large tasks, QCoRM handles them efficiently.

6 Related Work

RMs can be viewed as predicate-labelled Mealy/Moore machines in classical finite-automata (FA) theory [15]. However, RM semantics depend on structured observation guards and transition rewards, so classical deterministic FA minimisation is not directly applicable. This motivates agenda RMs, which augment states with structured information about task progression. Coupled RMs yield a non-deterministic FA-like execution semantics in which the agent may simultaneously occupy a set of coupled states.

Beyond the RMs discussed in our paper, a separate line of work uses automata for task decomposition with the automaton structure inferred from experience rather than given by the user. First, Furelos-

Blanco et al. [12] introduced ISA—a method for learning subgoal automata with binary rewards (0 or 1) from traces. Next, Furelos-Blanco et al. [13] introduced hierarchies of RMs (HRMs) which are learnt from traces. Compared to the RMs in our work, HRMs only support labelling edges with real-valued rewards rather than reward functions, and learning with HRMs can be globally sub-optimal because each automaton is learnt in isolation. More recently, Ardon et al. [3] introduced first-order RMs (FORMs), where first-order logic is used for compact task specifications. However, in contrast to the RMs discussed in our paper, FORMs support only real-valued rewards, and, for any number of unordered subtasks, there is only one FORM state expressed by a universally quantified FO atom, thus providing no guidance for unordered-subtask completion.

Here, we compare our QCoRM algorithm with task-decomposition RL methods for long-horizon tasks with unordered subtasks. RM-based RL decomposes tasks into symbolically defined subtasks. By contrast, methods that discover low-level skills/options [4, 22] or do not accept symbolic specifications [28, 25] are outside this comparison but could complement RMs. Learning a policy per subtask is well-established [27]. Some notable works include options [34], policy sketches [2], taskable RL [17], and hierarchical policies [8]. These methods typically optimise Markovian returns and do not, by default, provide semantics for non-Markovian rewards. Supporting history-dependent objectives generally requires state augmentation or explicit automata [32]. In addition to RMs, some notable works include R-AVI [20], DiRL [19], and LSTS [31]. Both DiRL and LSTS translate SPECTRL tasks into abstract graphs. A high-level policy over graph transitions is then found via Dijkstra's algorithm with costs based on transition success probabilities between state-space regions.

Likewise, QCoRM uses the RM structure to guide high-level decision making. However, QCoRM converges to optimal solutions via shaping low-level policies by the global episode-length signal. In contrast, DiRL's local policies optimise only for individual abstract transitions, so the composed high-level policy is not guaranteed to be globally optimal. Moreover, DiRL learns one low-level policy at a time, whereas QCoRM learns a low-level policy for each subtask in all coupled RM states in parallel. This distinction becomes crucial when tasks contain many unordered subtasks.

7 Limitations and Future Work

Here, we briefly summarise limitations of the presented work. First, the exponential representation of coupled RMs, and consequently of the corresponding η tables, leaves an exponential-memory bottleneck. As every completion order must remain representable to find an optimal one, avoiding this residual exponential while retaining global optimality is an open question. Second, all tested domains are deterministic. Even though QCoRM is well-defined in stochastic environments because $\eta(u)$ and Theorem 3 are formulated in terms of expected values, the transfer of the global optimality guarantee to stochastic dynamics remains an open question. Third, the initial position of the agent is fixed in QCoRM. Supporting varied initial positions would require conditioning η on the initial environment state. Fourth, QCoRM requires assumptions to be met. Numeric variables must be discrete and finite (Assumption 1), and features must be observable (Assumption 3); otherwise, partially observable RMs are needed. If numeric variables could increase, violating Assumption 2, Proposition 1 would no longer hold, and a fourth feature value for increases would introduce cycles in the unfolded RM, breaking the depth-based annotation of Proposition 2. Further, only one subtask can be completed per step (Assumption 6), and groups of unordered subtasks are completed sequentially. Supporting simultaneous completions would require fanning out the update (4) over the completed subset. Finally, learning with coupled RMs could be extended beyond Q-learning in future work.

8 Conclusion

To tackle long-horizon tasks with unordered subtasks, we propose Q-learning with coupled RMs (QCoRM), a new task-decomposition algorithm that scales to more unordered subtasks than baseline methods in tabular and continuous domains. Notably, the optimality guarantee of tabular Q-learning is preserved in QCoRM. To enable QCoRM, we introduce agenda and coupled RMs: agenda RMs identify mergeable task-progression states, and coupled RMs enable subtask-level policy sharing. In addition, we introduce numeric RMs that assist the user with compact task representation. Our experiments in the Delivery, Office, Water, and HalfCheetah domains show that QCoRM scales better than baseline algorithms to more unordered subtasks in terms of both convergence speed and runtime.

Acknowledgments and Disclosure of Funding

This work was supported by Ericsson Research and the Wallenberg AI, Autonomous Systems, and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725. The work of N. Pappas has been supported in part by ELLIIT and the European Union (6G-LEADER and 101192080). We express gratitude to the reviewers for their invaluable feedback and comments that helped to strengthen the paper; we also thank Arnaud Lequen for fruitful discussions.

References

[1] S. Agrawal and N. Goyal. Further optimal regret bounds for Thompson sampling. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, pages 99–107, 2013.

[2] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning*, pages 166–175. PMLR, 2017.

[3] L. Ardon, D. Furelos-Blanco, R. Parac, and A. Russo. FORM: Learning expressive and transferable first-order logic reward machines. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, pages 142–151, 2025.

[4] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[5] A. Camacho, R. Toro Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 6065–6073, 2019.

[6] W. Cui and W. Yu. Reinforcement learning with non-cumulative objective. *IEEE Transactions on Machine Learning in Communications and Networking*, 1:124–137, 2023.

[7] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[8] D. Drexler, J. Seipp, and H. Geffner. Learning hierarchical policies by iteratively reducing the width of sketch rules. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, pages 208–218, 2023.

[9] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. Challenges of real-world reinforcement learning: Definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

[10] J. Eschmann. Reward function design in reinforcement learning. *Reinforcement Learning Algorithms: Analysis and Applications*, pages 25–33, 2021.

[11] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[12] D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. Induction and exploitation of subgoal automata for reinforcement learning. *Journal of Artificial Intelligence Research*, 70: 1031–1116, 2021.

[13] D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. Hierarchies of reward machines. In *Proceedings of the 40th International Conference on Machine Learning*, pages 10494–10541. PMLR, 2023.

[14] A. Garivier and E. Moulines. On upper-confidence bound policies for switching bandit problems. In *Proceedings of the 22nd International Conference on Algorithmic Learning Theory*, pages 174–188, 2011.

[15] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2nd edition, 2001.

[17] L. Illanes, X. Yan, R. Toro Icarte, and S. A. McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 540–550, 2020.

[18] K. Jothimurugan, R. Alur, and O. Bastani. A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

[19] K. Jothimurugan, S. Bansal, O. Bastani, and R. Alur. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34:10026–10039, 2021.

[20] K. Jothimurugan, O. Bastani, and R. Alur. Abstract value iteration for hierarchical reinforcement learning. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, pages 1162–1170. PMLR, 2021.

[21] A. Karpathy. REINFORCEjs: WaterWorld demo. `https://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html`, 2015. Accessed: 2025-07-31.

[22] M. Klissarov and M. C. Machado. Deep Laplacian-based options for temporally-extended exploration. In *Proceedings of the 40th International Conference on Machine Learning*, pages 17198–17217. PMLR, 2023.

[23] K. Levina. Q-learning with coupled reward machines (QCoRM): Code and experiment data. Zenodo, 2026. URL `https://doi.org/10.5281/zenodo.23567987`.

[24] A. C. Li, Z. Chen, T. Q. Klassen, P. Vaezipoor, R. Toro Icarte, and S. A. McIlraith. Reward machines for deep RL in noisy and uncertain environments. *Advances in Neural Information Processing Systems*, 37:110341–110368, 2024.

[25] J. A. Mendez, H. van Seijen, and E. Eaton. Modular lifelong reinforcement learning via neural composition. In *Proceedings of the 10th International Conference on Learning Representations*, 2022.

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

[27] A. Mohan, A. Zhang, and M. Lindauer. Structure in deep reinforcement learning: A survey and open problems. *Journal of Artificial Intelligence Research*, 79:1167–1236, 2024.

[28] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[29] C. Neary, Z. Xu, B. Wu, and U. Topcu. Reward machines for cooperative multi-agent reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems*, pages 934–942, 2021.

[30] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021.

[31] Y. Shukla, T. Burman, A. N. Kulkarni, R. Wright, A. Velasquez, and J. Sinapov. Logical specifications-guided dynamic task sampling for reinforcement learning agents. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 532–540, 2024.

[32] J. Skalse and A. Abate. On the limitations of Markovian rewards to express multi-objective, risk-sensitive, and modal tasks. In *Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence*, pages 1974–1984. PMLR, 2023.

[33] S. Srivastava, S. Zilberstein, N. Immerman, and H. Geffner. Qualitative numeric planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pages 1010–1016, 2011.

[34] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[35] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[16] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.

[36] M. Towers, A. Kwiatkowski, J. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, J. Terry, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A standard interface for reinforcement learning environments. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-4916.

[37] H. Unniyankal, F. Belardinelli, A. Ferrando, and V. Malvone. RMLGym: A formal reward machine framework for reinforcement learning. In *WOA 2023: 24th Workshop From Objects to Agents*, CEUR Workshop Proceedings, 2023.

[38] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[39] C. K. Verginis, C. Koprulu, S. Chinchali, and U. Topcu. Joint learning of reward machines and policies in environments with partially known semantics. *Artificial Intelligence*, 333:104146, 2024.

[40] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

A Proof of Proposition 1

Proposition 1 (Proposition 1, restated)**.** *Under Assumption 2, domain $\mathcal{D}_{p_w}$ fully captures possible changes between the current and previous values of w: w_t and w_{t-1}.*

Proof. Since w cannot increase (Assumption 2) and is lower-bounded by w_ℓ, there are four possible relations between w_t, w_{t-1}, and w_ℓ: (1) $w_{t-1} > w_t > w_\ell$ maps to $p_w{\downarrow}$; (2) $w_{t-1} > w_t = w_\ell$ and (3) $w_{t-1} = w_t = w_\ell$ map to $p_{w\underline{\downarrow}}$; and (4) $w_{t-1} = w_t > w_\ell$ maps to $p_w!$. The mapping is thus complete. □

B Proof of Proposition 2

Proposition 2 (Proposition 2, restated)**.** *Under Assumptions 2 and 4–6, the annotation scheme in an agenda RM uniquely identifies RM states with respect to task progression.*

Proof. Let RM states u and u' be reached after the agent completes subtask sequences h and h', respectively, let X and X' be the corresponding sets of possible next subtasks, and let $\Xi(u) = \langle d, (T_w)_{w\in W}, x\rangle$ and $\Xi(u') = \langle d', (T'_w)_{w\in W}, x'\rangle$. We show that $\Xi(u) = \Xi(u')$ if and only if $\langle X, |h|, \Sigma_h\rangle = \langle X', |h'|, \Sigma_{h'}\rangle$, i.e. u and u' correspond to the same task progression (Definition 4). By Assumption 6, each transition between distinct RM states completes one subtask, and by Assumption 2, no transition undoes a completed subtask, so $d = |h|$ and $d' = |h'|$.

Let $\langle X, |h|, \Sigma_h \rangle = \langle X', |h'|, \Sigma_{h'} \rangle$. Hence, $d = d'$. The sequences in Σ_h contain all and only the subtasks that remain after h, so h and h' leave the same agenda $T_w = T'_w$ for every $w \in W$. The current objective is fixed by the set X of possible next subtasks: if X consists of one Boolean feature, x is this feature, and otherwise, since groups of unordered subtasks are completed one after another, X is the agenda of a single group, i.e. $x = X = T_w$ for some $w \in W$. Thus, $x = x'$ and $\Xi(u) = \Xi(u')$.

Conversely, let $\Xi(u) = \Xi(u')$. Then $|h| = |h'|$ because $d = d'$. Each RM state has a single current objective, either a Boolean feature or the agenda T_w of some $w \in W$; therefore, $X = X'$. Next, since both states have the same agenda T_w for every $w \in W$, h and h' have completed the same unordered subtasks of each group, possibly in different orders. Moreover, as Boolean subtasks are ordered and stay outside the agendas by Assumption 5, h and h' have completed the same Boolean subtasks. Hence, after $|h|$ subtasks, the same objectives remain, and, as groups of unordered subtasks are completed one after another, they remain in the same order, so $\Sigma_h = \Sigma_{h'}$. Overall, $\langle X, |h|, \Sigma_h \rangle = \langle X', |h'|, \Sigma_{h'} \rangle$. □

C Proof of Theorem 3

Theorem 3 (Theorem 3, restated)**.** *The final reward for completing any subtask τ preserves the global optimality guarantee of tabular Q-learning if it is defined as*

$$R(K, K^\tau) = \begin{cases} \check{R} & \text{if } \delta K^\tau = 0, \\ \gamma^{\delta K^\tau + 1} \check{R} + \frac{\gamma^{1-K^\tau} - \gamma^{\delta K^\tau}}{1-\gamma} r_{mi} & \text{otherwise}, \end{cases} \tag{5}$$

where K is the episode length, K^τ is the τ execution duration in timesteps, $\check{R} \in \mathbb{R}_{>0}$, $\delta K^\tau = \max\{0, \bar{K}^\tau_{op} - K^\tau, K - \bar{K}_{mi}\}$, $\bar{K}^\tau_{op}$ is the expected τ duration under a globally optimal policy (updated during the high-level exploitation as $\bar{K}^\tau_{op} \xleftarrow{\alpha_K} K^\tau$), $\bar{K}_{mi}$ is the minimum episode length (updated during the high-level exploitation as $\bar{K}_{mi} \xleftarrow{\alpha_K} K$ if $K < \bar{K}_{mi}$), α_K is the learning rate, and $r_{mi} \leq 0$ is the minimum immediate reward.

Proof. Part 1: Agent prefers globally optimal policies under given $\bar{K}_{mi}$ and $\bar{K}^\tau_{op}$. Consider an arbitrary subtask τ that is part of some complete policy for the entire task. The discounted return for completing τ with any *locally sub-optimal* policy should exceed that of any *locally optimal* policy if the former contributes to a *globally optimal* solution. Let π^τ_1 be a locally optimal but globally sub-optimal policy with the expected local duration $\mathbb{E}[K^\tau_1] = \bar{K}^\tau_1$ and induced episode length $\bar{K}_1 > \bar{K}_{\text{op}}$. Let π^τ_{op} be a locally sub-optimal but globally optimal policy with the expected local duration $\bar{K}^\tau_{\text{op}}$ and induced episode length $\bar{K}_{\text{op}}$. By assumption, $\bar{K}^\tau_{\text{op}} > \bar{K}^\tau_1$.

If the final reward were equal to constant $\check{R}$, the agent would prefer π^τ_1 over π^τ_{op}, since π^τ_1 is locally optimal and hence yields a higher expected return (1): $\check{Q}^\tau_1 > \check{Q}^\tau_{\text{op}}$,

$$\mathbb{E}_{\pi^\tau_1}[\check{R}\gamma^{K^\tau_1 - 1} + \sum_{k_1=0}^{K^\tau_1 - 2} \gamma^{k_1} r^\tau_{k_1+1}] > \mathbb{E}_{\pi^\tau_{\text{op}}}[\check{R}\gamma^{K^\tau_{\text{op}} - 1} + \sum_{k_{\text{op}}=0}^{K^\tau_{\text{op}} - 2} \gamma^{k_{\text{op}}} r^\tau_{k_{\text{op}}+1}].$$

For readability, we omit the time index t.

To prevent convergence to a globally sub-optimal policy, the final reward should depend on K^τ_1, and the returns become

$$Q^\tau_1 = \mathbb{E}_{\pi^\tau_1}[R(K^\tau_1)\gamma^{K^\tau_1 - 1} + \sum_{k_1=0}^{K^\tau_1 - 2} \gamma^{k_1} r^\tau_{k_1+1}]$$

$$Q^\tau_{\text{op}} = \check{Q}^\tau_{\text{op}}.$$

For global optimality, $Q^\tau_{\text{op}} > Q^\tau_1$, so the reward must satisfy

$$R(\bar{K}^\tau_1) < \check{R} - \gamma^{1 - \bar{K}^\tau_1}(\check{Q}^\tau_1 - \check{Q}^\tau_{\text{op}}).$$

Because in many non-cumulative tasks, agents typically receive non-positive intermediate rewards to avoid non-optimal behaviours [6], we assume that immediate rewards are bounded between $r_{\text{mi}} \leq 0$

and $r_{\text{ma}} \leq 0$. We then obtain

$$\check{Q}_1^\tau - \check{Q}_{\text{op}}^\tau \leq \check{R}(\gamma^{\bar{K}_1^\tau - 1} - \gamma^{\bar{K}_{\text{op}}^\tau - 1}) - \frac{1 - \gamma^{\bar{K}_{\text{op}}^\tau - 1}}{1 - \gamma} r_{\text{mi}}.$$

Rearranging yields the bound on the reward

$$R(\bar{K}_1^\tau) < \gamma^{\bar{K}_{\text{op}}^\tau - \bar{K}_1^\tau} \check{R} + \frac{\gamma^{1 - \bar{K}_1^\tau} - \gamma^{\bar{K}_{\text{op}}^\tau - \bar{K}_1^\tau}}{1 - \gamma} r_{\text{mi}}.$$

Since $\gamma^{\bar{K}_{\text{op}}^\tau - \bar{K}_1^\tau} > \gamma^{\max\{0, \bar{K}_{\text{op}}^\tau - \bar{K}_1^\tau, \bar{K}_1 - \bar{K}_{\text{mi}}\} + 1}$, as $\gamma < 1$, expression (5) satisfies the obtained bound for any local duration $K^\tau < \bar{K}_{\text{op}}^\tau$ or any realised episode length $K > \bar{K}_{\text{mi}}$. The reward (5) uses K and K^τ available from each episode. The return ordering holds almost surely because Q-learning averages these into the expected Q-values [40]. The result is valid for stationary $\bar{K}_{\text{mi}}$ and $\bar{K}_{\text{op}}^\tau$, but they are updated online during the high-level exploitation:

$$\bar{K}_{\text{mi}} \xleftarrow{\alpha_K} K \text{ if } K < \bar{K}_{\text{mi}};$$
$$\bar{K}_{\text{op}}^\tau \xleftarrow{\alpha_K} K^\tau.$$

Part 2: $\bar{K}_{mi}$ and $\bar{K}_{op}^\tau$ converge to $\bar{K}_{mi}^*$ and $\bar{K}_{op}^{*\tau}$, respectively. To push the optimal episode length to minimum, $\bar{K}_{\text{mi}}$ is updated when $K < \bar{K}_{\text{mi}}$, so the sequence $\{\bar{K}_{\text{mi}}\}$ is non-increasing and bounded below by $\bar{K}_{\text{mi}}^* > 0$. At each fixed $\bar{K}_{\text{mi}}$ and $\bar{K}_{\text{op}}^\tau$, the reward (5) is stationary; by Watkins and Dayan [40], Q-learning contracts to optimal Q^*-values for each low-level policy. By Part 1, each converged low-level policy achieves $\delta K^\tau = 0$, so the reward is the constant $\check{R}$, independent of episode length. Then, with the reward fixed at $\check{R}$, the Bellman contraction with $\gamma < 1$ attempts to minimise each subtask duration K^τ, since $\gamma^{K^\tau - 1}\check{R}$ is strictly decreasing in K^τ. If the minimised local durations collectively do not decrease the global episode length, the reward punishes the agent, pointing it to $\delta K^\tau = 0$. However, if the summed durations of local policies within a global policy are reduced, $\bar{K}_{\text{mi}}$ is reduced, changing the reward expression. Since no episode shorter than $\bar{K}_{\text{mi}}^*$ exists, the sequence stabilises at $\bar{K}_{\text{mi}}^*$, and the globally optimal Q^*-values follow. The expected local durations $\bar{K}_{\text{op}}^{*\tau}$ are then aligned with the optimal episode length. Therefore, the reward (5) preserves global optimality for any K^τ and τ. Part 2 thus mirrors policy iteration [34], with the convergence of the low-level Q-values under fixed $\bar{K}_{\text{mi}}$ and $\bar{K}_{\text{op}}^\tau$ in the role of policy evaluation and the update of $\bar{K}_{\text{mi}}$ and $\bar{K}_{\text{op}}^\tau$ from the resulting episodes in the role of policy improvement, repeated until the estimates no longer change. □

Example 1 (Reward (5) in a two-ball Water task)**.** Consider a simplified Water task in which the agent must touch balls 1 and 2 (subtasks τ_1 and τ_2) in any order, with $\check{R} = 1$, $r_{\text{mi}} = 0$, and $\gamma = 0.9$. The shortest episode touches ball 2 first and takes 5 steps: the agent decelerates before ball 2, touching it after 4 steps, and is then positioned to reach ball 1 in 1 step. The locally fastest policy for τ_2 touches ball 2 after 3 steps but leaves the agent badly positioned, so reaching ball 1 takes 3 more steps and the episode takes 6 steps. That is, decelerating costs one step before ball 2 and saves two steps afterwards. With $r_{\text{mi}} = 0$, the reward (5) reduces to $R = \check{R}$ if $\delta K^\tau = 0$ and to $R = \gamma^{\delta K^\tau + 1}\check{R}$ otherwise. Under the converged estimates $\bar{K}_{\text{mi}} = 5$ and $\bar{K}_{\text{op}}^{\tau_2} = 4$, Table 1 compares both policies for τ_2. Without shaping, the locally fastest policy would receive the higher return, $\gamma^2 = 0.81 > \gamma^3 = 0.729$, and would be preferred. With the shaped reward, its local duration is one step shorter than $\bar{K}_{\text{op}}^{\tau_2}$ and its episode one step longer than $\bar{K}_{\text{mi}}$, so $\delta K^{\tau_2} = 1$ and its return drops to $\gamma^4 = 0.6561$. The agent therefore keeps the globally optimal policy that decelerates before ball 2.

Table 1: Reward (5) for two policies for subtask τ_2 in Example 1, with $\bar{K}_{\text{mi}} = 5$ and $\bar{K}_{\text{op}}^{\tau_2} = 4$.

Policy for τ_2	K^{τ_2}	K	δK^{τ_2}	R	Return $R\gamma^{K^{\tau_2}-1}$
Globally optimal	4	5	$\max\{0,0,0\} = 0$	$\check{R} = 1$	$\gamma^3 = 0.729$
Locally fastest	3	6	$\max\{0,1,1\} = 1$	$\gamma^2 = 0.81$	$\gamma^4 = 0.6561$

D Hyperparameters

For tabular Q-learning, the learning rate $\alpha = 0.1$, discount factor $\gamma = 0.9$, and exploration parameter $\epsilon = 0.1$. For DDQN, $\alpha = 10^{-4}$, $\gamma = 0.9$, soft update parameter $\rho = 0.1$, and ϵ decays linearly from 1 to 0.1 over the first 1% of training steps. The replay buffer and batch sizes are 50K and 32, respectively, for QRM and are scaled up by $\lceil N/2 \rceil$ for QCoRM, where N is the number of unordered subtasks, and by $|U|$ for HRM and CRM. This scaling keeps the buffer and batch sizes proportional to the number of experiences added per environment step: QCoRM adds one experience for each current coupled state, whose number decreases from N to 1 as the subtasks are completed, giving $(N+1)/2$ on average, whereas CRM and HRM add experiences for all $|U|$ RM states. The RM-exploration parameter ξ decays from 1 to 0.1 in increments of 0.001. For TD3, $\alpha = 3 \times 10^{-4}$, $\gamma = 0.99$, $\rho = 0.005$, buffer size is 10^6, and batch size is 256, both scaled up by $\lceil N/2 \rceil$ for QCoRM. The DDQN networks have three layers with 512 hidden units each, and the TD3 actor and critics have two layers with 400 and 300 hidden units; the training frequency and the number of gradient steps are 1. In all domains, the episode-length learning rate α_K is 0.3, and $\alpha_\eta = \min\{0.005\,|U'|, 0.05\}$ grows with the number of subtasks $|U'|$ to account for the coupled updates. All other parameters use the default setting.

E QCoRM Pseudocode

Algorithm 1 summarises QCoRM, tying together the high-level subtask selection, parallel low-level updates over coupled states, and reshaped replay of valuable experiences at the end of each exploitation episode.

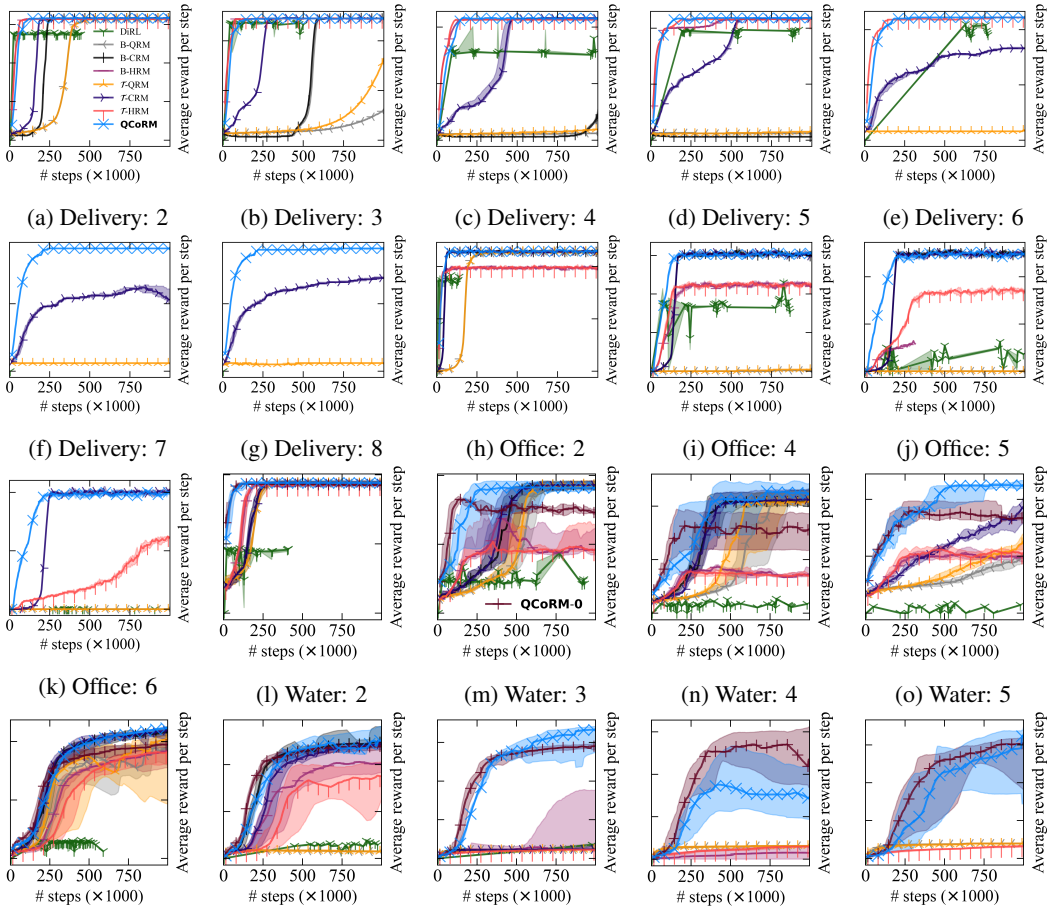
Algorithm 1 QCoRM: Q-learning with a coupled RM

Require: coupled RM with states $u \in U$ and subtasks $\tau \in U'$; rates $\alpha, \alpha_\eta, \alpha_K$; discount γ; explorations ϵ, ξ

1: initialise Q-tables; $\eta(u) \leftarrow 1$; $\bar{K}_{\text{mi}}, \bar{K}^{\tau}_{\text{op}} \leftarrow$ first observed K, K^{τ}
2: **for** each episode **do**
3: sample *exploit* $\sim \text{Bernoulli}(1-\xi)$; $\mathcal{B} \leftarrow \emptyset$; $K \leftarrow 0$
4: read coupled states $\{u_k\}_{k=1}^{m}$ with subtasks τ_k; $K_0 \leftarrow K$ {$m = 1$ for a Boolean objective}
5: select subtask τ: $\arg\min_k \eta(u_k)$ if *exploit*, else randomly {high-level policy}
6: **while** episode not terminated **do**
7: $a \leftarrow \epsilon$-greedy on $Q(\langle s, \tau \rangle, \cdot)$; execute a, observe s'; $K \leftarrow K+1$
8: **for all** coupled subtasks τ_k **do**
9: **if** τ_k completed **then**
10: append $(\langle s, \tau_k \rangle, a, K^{\tau_k})$ to $\mathcal{B}$, where $K^{\tau_k} = K - K_0$ {valuable exp.}
11: $K_0 \leftarrow K$
12: **else**
13: update $Q(\langle s, \tau_k \rangle, a)$ with Eq. (3)
14: **end if**
15: **end for**
16: on completion of any τ_k, repeat lines 4–5 for the next RM state
17: **end while**
18: **if** *exploit* **and** not failed episode **then**
19: if $K < \bar{K}_{\text{mi}}$: $\bar{K}_{\text{mi}} \xleftarrow{\alpha_K} K$
20: $\bar{K}^{\tau'}_{\text{op}} \xleftarrow{\alpha_K} K^{\tau'}$ for each completed subtask τ'
21: **for all** $(\langle s, \tau' \rangle, a, K^{\tau'}) \in \mathcal{B}$ **do**
22: reshape $R \leftarrow R(K, K^{\tau'})$ by Eq. (5)
23: update $Q(\langle s, \tau' \rangle, a)$ by Eq. (4)
24: **end for**
25: **end if**
26: $\eta(u_k) \xleftarrow{\alpha_\eta} K_{u_k \to \text{goal}}$ for each u_k with completed τ_k {high-level cost update}
27: **end for**

F Results for All Numbers of Unordered Subtasks

Fig. 3 shows the results of QCoRM and the baselines for all tested numbers of unordered subtasks. In Delivery, QCoRM and $\mathcal{T}$-HRM converge fastest for up to six boxes, followed by the CRM methods, which converge faster with the agenda RMs than with the Boolean RMs, and for seven and eight boxes, only QCoRM converges. In Office, QCoRM converges fastest for all numbers of offices, closely followed by the CRM methods, whereas the HRM methods converge to sub-optimal policies and the QRM methods converge only for two and three offices. In Water, QCoRM converges fastest for all numbers of balls, while QCoRM-0 learns quickly but remains sub-optimal for three to five balls. In HalfCheetah, QCoRM performs on par with the CRM methods for two and three positions, and for four to six positions, only QCoRM and QCoRM-0 make progress. DiRL's final performance stays below that of QCoRM in all tasks.



(a) Delivery: 2 (b) Delivery: 3 (c) Delivery: 4 (d) Delivery: 5 (e) Delivery: 6

(f) Delivery: 7 (g) Delivery: 8 (h) Office: 2 (i) Office: 4 (j) Office: 5

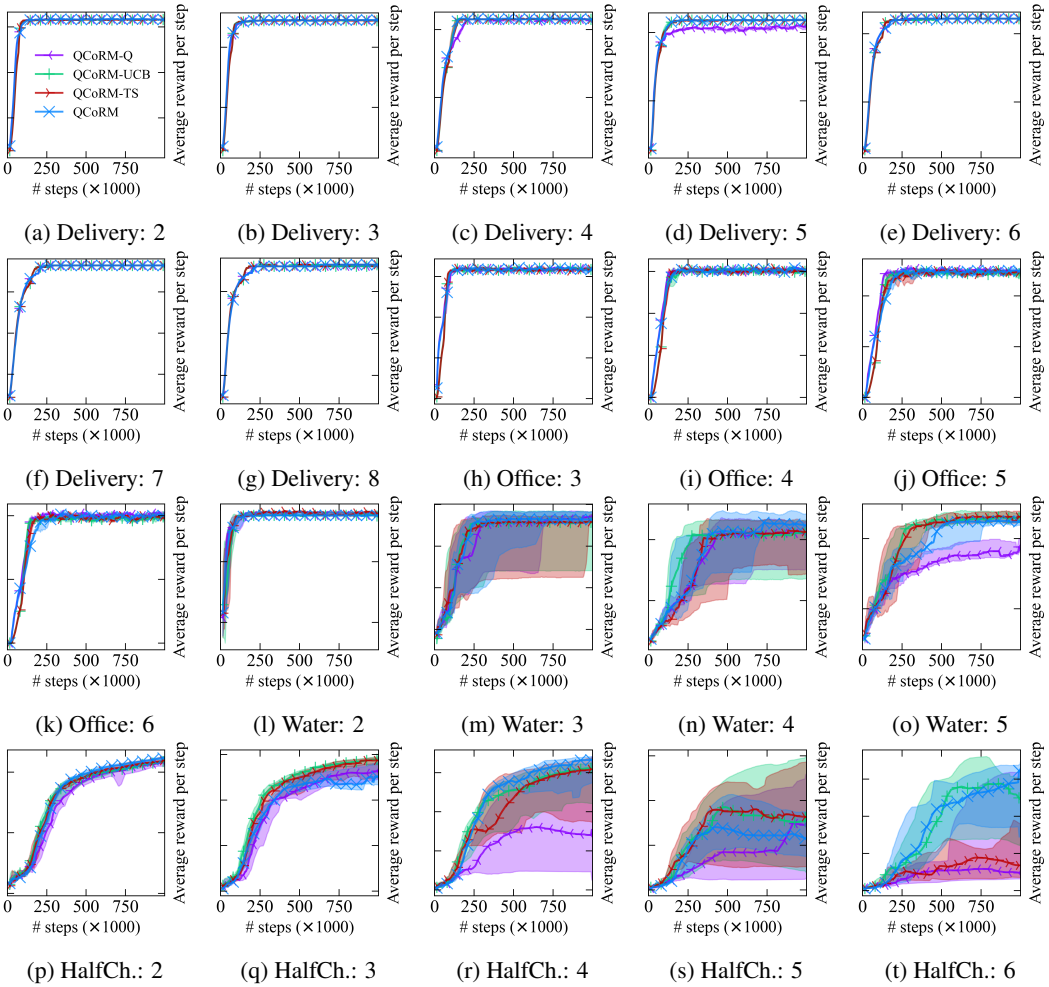
(k) Office: 6 (l) Water: 2 (m) Water: 3 (n) Water: 4 (o) Water: 5

(p) HalfCh.: 2 (q) HalfCh.: 3 (r) HalfCh.: 4 (s) HalfCh.: 5 (t) HalfCh.: 6

Figure 3: Results of QCoRM and the baselines for all numbers of unordered subtasks. Subcaptions give the number of unordered subtasks: boxes, offices, balls, or positions.

G High-Level Controllers in QCoRM

In addition to the multi-armed bandit with the constant step-size update (Section 4), we integrate three high-level controllers into QCoRM: discounted upper confidence bound (UCB), Thompson sampling (TS), and a high-level Q-function. Each controller selects the next subtask from the current coupled states $u_1, \ldots, u_m$ with subtasks $\tau_1, \ldots, \tau_m$ by minimising the cost estimate. For all controllers, the low-level policies and the reward (5) remain unchanged.



(a) Delivery: 2 (b) Delivery: 3 (c) Delivery: 4 (d) Delivery: 5 (e) Delivery: 6

(f) Delivery: 7 (g) Delivery: 8 (h) Office: 3 (i) Office: 4 (j) Office: 5

(k) Office: 6 (l) Water: 2 (m) Water: 3 (n) Water: 4 (o) Water: 5

(p) HalfCh.: 2 (q) HalfCh.: 3 (r) HalfCh.: 4 (s) HalfCh.: 5 (t) HalfCh.: 6

Figure 4: Comparison of the high-level controllers in QCoRM: the multi-armed bandit from Section 4 (QCoRM), discounted UCB (QCoRM-UCB), Thompson sampling (QCoRM-TS), and the high-level Q-function (QCoRM-Q). Subcaptions give the number of unordered subtasks: boxes, offices, balls, or positions. We report median performance across 10 seeds, with the shaded regions representing the 25th and 75th percentiles.

Discounted UCB. UCB selects arms by their estimated values and a confidence term decreasing with the number of selections, and its discounted variant [14] forgets old observations, which is useful with non-stationary costs drifting as the low-level policies improve. For each RM state u, the agent maintains a discounted count $n(u)$ and a discounted sum of observed costs $c(u)$. After each episode e, the agent discounts the statistics of all RM states $u \in U$ and adds the observations of the RM states visited during that episode, U_e:

$$n(u) \leftarrow \lambda n(u) + [\![u \in U_e]\!], \qquad c(u) \leftarrow \lambda c(u) + [\![u \in U_e]\!] K_{u \to \text{goal}},$$

where $\lambda \in (0, 1)$ is a discount factor. Discounting the unvisited RM states enlarges their confidence terms over time, allowing re-exploration of possibly outdated estimates. The agent then selects the subtask of the coupled state

$$u^* = \arg\min_k \left(\frac{c(u_k)}{n(u_k)} - \beta \sqrt{\frac{\ln\left(1 + \sum_{j=1}^{m} n(u_j)\right)}{n(u_k)}} \right),$$

where $\beta > 0$ scales the confidence term. Exploration is part of the selection rule, so the RM-exploration probability ξ is not used for subtask selection.

Thompson sampling. TS selects arms by sampling from a distribution over their costs. We use Gaussian sampling distributions [1] with the same discounted statistics as in UCB. For each coupled state u_k, the agent samples

$$\hat{\eta}(u_k) \sim \mathcal{N}\big(c(u_k)/n(u_k),\, \sigma(u_k)^2\big), \qquad \sigma(u_k) = \min\{\sigma_{\max}, 1/\sqrt{n(u_k)}\},$$

and selects the subtask of the coupled state with the lowest sample. As in UCB, discounting widens the sampling distributions of RM states that were not visited, and ξ is not used for subtask selection.

High-level Q-function. Instead of averaging the observed steps-to-go, this controller learns η by bootstrapping, as in Q-learning [40] over the coupled RM, where the choice among coupled states plays the role of an action and subtask durations play the role of costs. After each episode, the agent updates every RM state u_e from which it transitioned, in reverse order of the visits:

$$\eta(u_e) \xleftarrow{\alpha_Q} K^{\tau_e} + \min_{u' \in C(u_e)} \eta(u'),$$

where α_Q is a learning rate, K^{τ_e} is the duration of subtask τ_e completed from u_e, $C(u_e)$ is the set of coupled states the agent entered after leaving u_e, and the minimum is zero if u_e transitions to a goal state. Updating from the goal towards the initial state propagates the costs along the whole completion path in a single episode. This controller explores with probability ξ in the same way as the bandit in Section 4.

In all domains, we use $\beta = 0.1$, $\sigma_{\max} = 1$, $\alpha_Q = 0.1$, and $\lambda = 0.99$.

Results. Fig. 4 compares the four controllers in all domains with the same setup as in the main experiments. In the tabular domains, all controllers converge to the same performance at similar rates; only the high-level Q-function converges more slowly in Delivery with four and five boxes (Figs. 4c and 4d). In the continuous domains, the bandit, UCB, and TS perform comparably without a consistent winner, except for HalfCheetah with six positions, where TS seems to degrade (Fig. 4t). The high-level Q-function seems to be the weakest high-level controller likely because bootstrapping propagates the errors of the successor estimates while the subtask durations are still changing. Overall, the bandit, UCB, and TS controllers yield comparable performance, so QCoRM is not sensitive to the choice among them.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the contributions: Q-learning with coupled reward machines (QCoRM) with a global optimality guarantee in tabular settings, agenda and coupled RMs that enable QCoRM, and numeric RMs for compact task specification. These contributions are developed with theoretical results and evaluated experimentally in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: Section 7 discusses the exponential memory of coupled RMs, the deterministic experimental domains, the fixed initial position of the agent, the assumptions of QCoRM and the consequences of violating them, and extensions beyond Q-learning.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Propositions 1 and 2 and Theorem 3 are numbered and stated together with the assumptions they rely on, and their complete proofs are given in Appendices A, B, and C, which are cross-referenced from the main text.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

Justification: Section 4 describes QCoRM with its update equations, and Appendix E summarises it in Algorithm 1. Section 5 describes all domains, baselines, and the evaluation protocol, Appendix D lists all hyperparameters, and the code is publicly available.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Section 5 links to the code and experiment data on Zenodo.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 describes the domains, baselines, reward settings, and evaluation protocol; Appendix D lists the hyperparameters for tabular Q-learning, DDQN, and TD3, and Appendix G lists those of the alternative controllers.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Section 5 reports median performance across 10 seeds with shaded regions in all plots representing the 25th and 75th percentiles.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

Justification: Section 5 states that experiments ran on Intel Xeon Gold 6130 CPUs using a single core with 16 GiB of memory for tabular domains and 32 GiB for continuous domains, with each run limited to 10^6 steps and 40 hours.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper presents foundational reinforcement learning research; no human subjects are involved, and no sensitive data are collected.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This is foundational research on reward machine formalisms and a Q-learning algorithm; there is no direct path to harmful applications; its main impact is the general benefit of improved sample efficiency in reinforcement learning.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper releases only the algorithm implementation and experiment data; no pre-trained models, image generators, or scraped datasets are involved.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All external code and environments used in the experiments are cited: the reward machine codebase [16], the DiRL code [19], Stable-Baselines3 [30] (MIT licence), Gymnasium [36] (MIT licence), MuJoCo [35] (Apache 2.0 licence), and the Water environment [21].

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

Justification: The QCoRM implementation is released on Zenodo together with a README that documents its use, and the algorithm is fully described in the paper.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects, so no IRB approval is required.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: No LLMs are used as part of the core methodology; the paper's contributions are reward machine formalisms and a Q-learning algorithm.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.