scientific reports

Check for updates

OPEN

Safe reinforcement learning under temporal logic with reward design and quantum action selection

Mingyu Cai[1], Shaoping Xiao[2]✉, Junchao Li[2] & Zhen Kan[3]

This paper proposes an advanced Reinforcement Learning (RL) method, incorporating reward-shaping, safety value functions, and a quantum action selection algorithm. The method is model-free and can synthesize a finite policy that maximizes the probability of satisfying a complex task. Although RL is a promising approach, it suffers from unsafe traps and sparse rewards and becomes impractical when applied to real-world problems. To improve safety during training, we introduce a concept of safety values, which results in a model-based adaptive scenario due to online updates of transition probabilities. On the other hand, a high-level complex task is usually formulated via formal languages, including Linear Temporal Logic (LTL). Another novelty of this work is using an Embedded Limit-Deterministic Generalized Büchi Automaton (E-LDGBA) to represent an LTL formula. The obtained deterministic policy can generalize the tasks over infinite and finite horizons. We design an automaton-based reward, and the theoretical analysis shows that an agent can accomplish task specifications with the maximum probability by following the optimal policy. Furthermore, a reward shaping process is developed to avoid sparse rewards and enforce the RL convergence while keeping the optimal policies invariant. In addition, inspired by quantum computing, we propose a quantum action selection algorithm to replace the existing ε-greedy algorithm for the balance of exploration and exploitation during learning. Simulations demonstrate how the proposed framework can achieve good performance by dramatically reducing the times to visit unsafe states while converging optimal policies.

Markov Decision Processes (MDPs) usually model motion planning subject to stochastic uncertainties and have been applied to represent many engineering systems[1]. On the other hand, complex high-level specifications can be expressed via formal languages[2], including Linear and Signal Temporal logic (LTL and STL). Specifically, automaton-assisted control synthesis for general MDPs has caught growing attention to achieve tasks formulated by LTL languages. When assuming an MDP model is thoroughly knowledgeable, a common LTL-based objective is to optimize the balance of maximizing the task satisfaction probability and reducing the total expected cost[3,4].

We are in the era of Artificial Intelligence (AI). Advanced AI techniques, including Machine Learning (ML) and Evolutionary Algorithms (EAs), have been utilized in various science and engineering disciplines. Many pioneer works have been done in materials science, complex systems, robotics, and more. As one of the ML's subsets, Reinforcement Learning (RL) doesn't require pre-gathered data for supervision. Indeed, RL is a sequential decision-making process, and the agent learns optimal control policies via gathering experience from the unknown environment[5] and sometimes considering uncertain MDPs.

In RL, safety means the agent avoids visiting undesirable states in the exploration process. However, although standard RL algorithms can ensure optimal convergence, those algorithms generally lack safe protection on what happens during the learning process. For example, a mobile robot is not allowed to reach the control room, but it may visit this room using epsilon policies during learning. Consequently, the agent doesn't intend to explore safely while learning the optimal policy to maximize the collected reward. Therefore, safe RL has attracted more attention, and various methods have been proposed[6]. However, most existing methods[7–10] either hold strong

[1]Department of Mechanical Engineering, Lehigh University, 113 Research Drive, Bethlehem, PA 18015, USA. [2]Department of Mechanical Engineering, University of Iowa, 3131 Seamans Center, Iowa City, IA 52242, USA. [3]Department of Automation, University of Science and Technology of China, 443 Huangshan Road, Hefei 230026, Anhui, China. ✉email: shaoping-xiao@uiowa.edu

assumptions about the dynamic models or only focus on minimizing the risk of violating a safety specification without considering complex high-level specifications.

There have been several proposed works on abstraction-based scenarios of MDPs in the past few years. For example, some research works employed LTL formulas to specify the instructions for a control agent to learn optimal strategies, i.e., optimal policies. Specifically, many works[11–15] designed automaton-based rewards so that model-free RL agents could find the optimal policies satisfying LTL specifications with probabilistic guarantees. However, none of them addresses the critical safety issues during training.

Li et al.[16] designed a robustness-based automaton and combined control barrier functions to facilitate learning, but the tasks introduced in[16] were considered over finite horizons only. A related work[17] utilized a Limit-Deterministic Generalized Büchi Automaton (LDGBA)[18] to elucidate LTL specifications for learning enhancement. It proposed a model-based safe padding technique to prevent the system from entering bad states. However, as shown in our previous study[15], directly applying LDGBA with purely positional policies, i.e., deterministic policies, might fail to achieve some tasks because there was no tracking record of accepting sets. An accepting set consists of automaton states that satisfy the acceptance condition of an LDGBA.

On the other hand, there is a growth of interest in quantum supremacy because a quantum computing algorithm, demonstrated on a quantum computer, offers significant speedup compared to the best possible algorithm on a classical computer. Moreover, the integration of machine learning and quantum computing[19], called Quantum Machine Learning (QML)[20], investigates how to encode classical data in quantum states and leverage superposition properties of quantum systems for solving some specific problems. Particularly, there is an exciting topic for researchers, Quantum Neural Networks (QNNs), including Quantum Convolutional Neural Networks (QCNNs).

Quantum neural networks are computational neural network models, mostly feed-forward networks[21], in which quantum bits in quantum neurons process and pass the information. In addition, inspired by Convolutional Neural Networks (CNNs), QCNNs were developed and evaluated for recognizing quantum states encoded from symmetry-protected topological phases[22]. A typical application of QML was image recognition[23] utilizing QCNN (or QNN) and Quantum Boolean Image Processing (QBIP)[24]. Furthermore, QNNs have been implemented in Deep Reinforcement Learning (DRL) to enhance learning outcomes[25].

Quantum computing, in general, is much faster than classical computing, such as solving optimization problems[26]. Therefore, we can intuitively expect that RL and quantum computing will join forces to make faster AI. Saggio *et al.*[27] utilized a quantum communication channel for an agent interacting with the environment to speed up the RL process. Their method generated a quantum state that was a superposition of rewarded and non-rewarded action sequences at each quantum epoch. After the environment flipped the sign of the winning action sequence, a quantum algorithm was applied to improve the chance of selecting the best action sequence. Such a hybrid AI has been shown to speed up the RL process by 60%. However, their approach was only applied to the Deterministic Strictly Epochal (DSE) learning scenarios.

In another work, Dong and co-workers[28] utilized MDP states (actions) in the conventional RL as eigenbases to generate the state (action) spaces in Quantum Reinforcement Learning (QRL) by superposition. The eigenbases, i.e., eigen states or eigen actions, serve as the orthogonal bases in a Hilbert space corresponding to the generated quantum system. To update the probability of eigen actions by Grover's algorithm, the collected reward, and state value functions are needed to determine the number of iterations. In addition, each eigen action state has a duplicated copy to prevent memory loss after selecting an action in their approach. Ganger and Hu[29] extended Dong's work[28] by using state-action value functions, also named Q values, instead of state value functions in QRL.

In this paper, we propose a few advanced techniques in RL for motion planning. One of our contributions is to extend our previous results[15] by presenting a provably correct reward design and developing the model-based safe padding. We encode LTL specifications over an infinite horizon into an Embedded LDGBA (E-LDGBA) that can record unvisited accepting sets to enable the application of deterministic policies. By using the shaping process for dense rewards, rigorous analysis shows that optimizing the expected return of the shaped reward scheme is the same as maximizing the satisfaction probability of LTL.

Secondly, assuming the ability of local observation, model-based padding can effectively add a "shield" to avoid the agent entering into sink components with a probabilistic bound and to maintain safety during the learning process. We propose a concept of safety value functions, including state and action safety values, to estimate an agent's probability of entering a safe state. Combining RL's conventional value functions and the proposed safety value functions will maximize the agent's safe protection and task satisfaction.

There are some other works considering safety issues in RL. For example, Fernandez-Gauna *et al.*[30] defined Undesirable Terminal States (UTS) as some of the terminal states associated with negative rewards. Once the agent reached one UTS, the corresponding constraints were violated, and the environment would be reset to its initial state. Another work[31] presented an approach for provably safe learning. This work introduced Justified Speculative Control (JSC) that combined verified runtime monitoring with RL. If the system was accurately modeled, only safe actions were taken. Otherwise, any available action would be selected randomly. Differing from those works, we propose safety value functions to quantify how well the agent will avoid unsafe states starting from the current state or taking the current action. The action selection depends on the conventional RL value function and the newly-introduced safety value function.

In addition, this work improves several aspects of the result of[17]. First, we employ a novel automaton structure that has been verified to accept the same language as LDGBA, i.e., E-LDGBA, to make up for the drawbacks of LDGBA. Then, we develop a potential function in a reward-shaping process to maintain a dense reward, and the obtained optimal policy remains to satisfy the tasks with maximum probability. In addition, by dividing the LTL into two parts, one of which defines the safe properties, we show that the model-based safe padding via safety values remains the original optimal convergence invariant.

Finally, we propose a quantum action selection algorithm in this paper, inspired by Grover's algorithm and quantum gate/measurement noises, to substitute the ε-greedy action selection in conventional RL. As a difference from the work[27], the quantum state, representing the action space in our method, is created by the superposition of available actions at the current MDP state instead of action sequences at each episode[27]. On the other hand, this quantum state is locally generated at each learning step, and no duplication is needed for global updates as in the works of[28] and[29]. It shall be noted that there is no "perfect" copy of an arbitrary unknown quantum state available, according to the no-cloning theorem[32].

The organization of this paper is described below. Section "Problem formulation" formulates MDP, RL, LTL, automata, and the problem definition. After introducing E-LDGBA, Section "Automaton-based reward design" describes an automaton-based reward design and a reward-shaping process. Section "Safety value functions" proposes safety value functions, and quantum action selection is described in Section "Quantum action selection". Then, two examples are included in Section "Simulations and discussions", followed by the conclusion and forthcoming works.

Problem formulation

Quantum computing. In classical computing, i.e., binary computing, a classical bit is a binary piece of information that can only take one of two possible values or states; for example, logic states 0 or 1. A qubit, a short name for a quantum bit, is the quantum equivalence of a classical bit. Differing from a classical bit, a qubit can represent one of two basic states or one possible combination. Given the most common basis, $|0\rangle$ and $|1\rangle$ that correspond to the logic states 0 and 1 for a classical bit, a qubit in its superposition state can be expressed as

$$|q_1\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle \tag{1}$$

where α_0 and α_1 are complex coefficients. When we measure a qubit in its superposition state, i.e., Equation (1), the qubit would collapse into a state of the basis, e.g., $|0\rangle$ or $|1\rangle$, with the probability of $|\alpha_0|^2$ or $|\alpha_1|^2$, respectively. In addition, $|\alpha_0|^2 + |\alpha_1|^2 = 1$ shall be satisfied.

Similar to Equation (1), a quantum state in an n-qubit system can be written as

$$|\psi_n\rangle = |q_1 q_2 \dots q_n\rangle = \sum_{k=0}^{2^n-1} \alpha_k |k\rangle \tag{2}$$

where $|k\rangle$ are basic states of the n-qubit system, and α_k are the corresponding complex coefficients satisfying $\sum_{k=0}^{2^n-1} |\alpha_k|^2 = 1$. When being measured, the n-qubit state will collapse into one of the basic states $|k\rangle$ with the probability of $|\alpha_k|^2$.

In quantum computing, a quantum register in the n-qubit system is utilized to carry a superposition of n-qubit basic states. Compared to an n-bit classical register that can store any one of 2^n possible numbers, an n-qubit register can store any combination of 2^n numbers. As the quantum version of a classical logic gate, a quantum gate operates on a quantum register that is usually initialized as $|00...0\rangle$ to evolve its state during the quantum computation.

One of the standard quantum single-qubit gates is the Hadamard gate (H), which results in a superposition of equal parts $|0\rangle$ and $|1\rangle$ when operating on either $|0\rangle$ or $|1\rangle$, as shown below.

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \quad H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \tag{3}$$

Indeed, the right-hand sides of Equation (3) are another set of basic states, named $|+\rangle$ and $|-\rangle$, which are also commonly used.

In addition, among the Pauli gates (X, Y, Z) and the Phase gates (S and T), the X gate is the quantum analog of the classical *NOT* gate with respect to $|0\rangle$ and $|1\rangle$, i.e., $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. In addition, a commonly used two-qubit gate is *CNOT* or *CX* gate, which provides the quantum equivalence of the classical *XOR* gate. *CNOT* gate stands for the controlled-not gate and is always applied on two qubits. If the first qubit (as the control one) is $|1\rangle$, then a *NOT* operation is applied to the second qubit (as the target one). Otherwise, the target qubit remains the same. For example,

$$CNOT|01\rangle = |01\rangle, \quad CNOT|11\rangle = |10\rangle \tag{4}$$

Markov decision process. Mathematically, an MDP, denoted by $\mathcal{M} = (S, A, p_S, s_0, \Pi, L)$, consists of a finite state space S and a finite action space A. In addition, $A(s)$ at state s denotes the available actions the agent can take at this state. The transition function, $p_S : S \times A \times S \to [0, 1]$, defines the probability of the agent moving from one state (s) to another (s') after it takes an action (a). Particularly, $\sum_{s'} p_S(s, a, s') = 1$ shall be satisfied. Usually, the transition probability p_S describes the motion uncertainties in an RL problem. Π is a set of atomic propositions. The labeling function, $L : S \to 2^\Pi$ where 2^Π is the power set of Π, assigns a subset of Π to each state. Sometimes, there exists an initial state $s_0 \in S$ with which the agent starts.

When an action function, $\xi : S \to A$, is deterministic, it outputs an action (a) at a given state (s). The MDP evolves gradually after performing action $\xi_i = a$ at step i ($i \geq 0$). Consequently, a sequence of actions $\boldsymbol{\xi} = \xi_0 \xi_1 \dots$ defines the control policy and generates a path $\boldsymbol{s} = s_0 s_1 s_2 \dots$ over $\mathcal{M}$. It shall be noted that the transition probability $p_S(s_i, a_i, s_{i+1})$ is positive for all i. For a stationary policy, we have $\xi_i = \xi$ for all i. On the other hand, if

$\xi_i = \xi(s_i)$ depends on the current state s_i only, the control policy is memoryless. Otherwise, the policy shall have a finite memory with the history of visited states, i.e., $\xi_i = \xi(..., s_{i-1}, s_i)$.

An MDP has a Markov property if the environment is fully observable and an agent carries out a simple go-to-goal task. Consequently, each decision-making relies on the current state only, so the control policy is memoryless. However, one of this study's focuses is developing a framework to handle complex tasks by introducing a product of MDP and LTL-induced automaton. The induced control policy is usually a finite memory one, i.e., the action selection depends on the current and past states.

Reinforcement learning. The environment in an RL problem can be expressed by an MDP as defined in Section "Markov decision process". An agent learns the policy via communicating with the environment, and learning is an iterative process. For example, Fig. 1 shows that the agent decides which action to take after identifying the present state. Once conducting the selected action, it receives feedback, i.e., the reward, and observes the next state for decision-making of the following action.

The reward function is one of the key elements in an RL problem. Generally, a reward function can be denoted by $\Lambda(s, a, s')$, which generates the reward to an agent after it takes an action (a) at the current state (s) and reaches another state (s'). In addition, a discount function $\gamma(s, a, s') \in [0, 1]$ is usually employed, and here defines the expected discounted return an agent can receive under policy $\boldsymbol{\xi}$ after starting from state s.

$$U^{\boldsymbol{\xi}}(s) = \mathbb{E}^{\boldsymbol{\xi}}\left[\sum_{i=0}^{\infty} \gamma^i(s_i, a_i, s_{i+1}) \cdot \Lambda(s_i, a_i, s_{i+1}) \middle| s_0 = s\right] \tag{5}$$

Equation (5) is also implied as the utility or the state value function generated from policy $\boldsymbol{\xi}$. In an RL problem, the state value function $U(s)$ quantifies how well the agent can reach a goal state over the long run, starting from state s. The learning objective is finding an optimal policy, $\boldsymbol{\xi}^* = \text{argmax}_{\boldsymbol{\xi}} U^{\boldsymbol{\xi}}(s)$, which can maximize the state value at each state. On the other hand, an optimal policy guides the agent in deciding on the best action to accomplish the task. When the environment is not fully known, i.e., the transition probability in its MDP model is unknown, model-free RL methods with tabular approaches are usually utilized for finite state and action spaces.

The methods in RL can be categorized as policy-based or value-based. The value-based RL methods directly solve and converge the value functions as the optimal ones. Q-learning[33] is one of them, and it solves action values or Q values instead of state values. A Q value is a function of state and action, i.e., $Q(s, a)$, represents the expected return an agent can collect under the current policy after taking action a at state s. Once the optimal action values are converged, we can obtain an optimal policy via the greedy action selection. It shall be noted that state values are related to action values by $U(s) = \max_a Q(s, a)$.

Since Q-learning doesn't require a state-to-state transition function, it is one of the model-free RL methods. When an RL problem has finite state/action spaces, a Q table is adopted in the naïve Q-learning method to store the value of every action at each state. Therefore, the best action can be determined at each state by searching for the highest Q value in the table. This method employs Monte Carlo simulations to converge Q values. The learning usually takes many episodes. Each episode consists of many steps or ends once the agent accomplishes the task. At each step, after taking an action (a), the agent moves from one state (s) to another (s'). Then, the Q value of action a at the current state s can be updated via the Bellman equation[5].

$$Q_{\text{new}}(s, a) = Q(s, a) + \alpha\left[\Lambda(s, a, s') + \gamma(s, a, s') \max_{a'} Q(s', a') - Q(s, a)\right] \tag{6}$$

where $\max_{a'} Q(s', a')$ represents the highest action value at state s'.

Equations (5) and (6) include generalized definitions of the reward function $\Lambda(s, a, s')$ and the discount factor $\gamma(s, a, s')$. Indeed, in a commonly used formulation[5,33], they are functions of the current state only, as $\Lambda(s)$ and $\gamma(s)$, respectively. In addition, a properly designed learning rate, α, is essential. If α is large, the convergence is fast but sometimes unstable. In addition, non-optimal value functions may be reached. On the other hand, although a smaller α can result in a smoother and more stable convergence procedure, the procedure may be

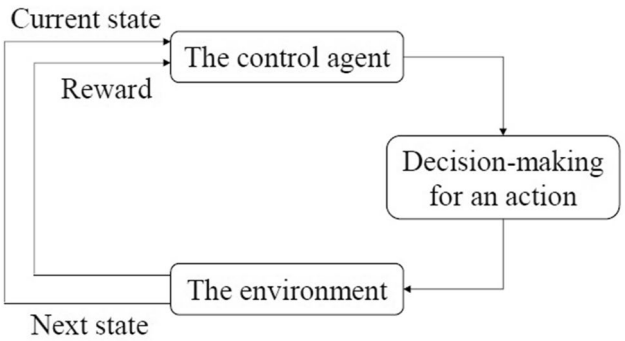

Figure 1. The agent interacts with the environment in a reinforcement learning problem.

slower. It is practical to employ an adaptive learning rate scheme to start with a large learning rate and decrease it with iterations.

Deep neural networks are usually employed to approximate value functions if the state and/or action spaces in an RL problem are large or continuous. The methods are then called Deep Reinforcement Learning (DRL) methods. The commonly used methods include Deep Q Network (DQN)[34], a variation of Q-learning, and Proximal Policy Optimization (PPO)[35], a policy-based method.

Linear temporal logic. Instead of simple go-to-goal tasks, this paper considers complex high-level tasks that LTL, a formal language, can describe. Specifically, we can identify and verify properties about how a world changes over time using LTL. The properties of interest, given high-level specifications of a system, include safety (i.e., anything terrible will never happen), liveness (i.e., something good will finally occur), and fairness (i.e., independent subsystems can make progress).

An LTL formula can be built on atomic propositions Π, some Boolean operators, including True, negation $\neg$, and conjunction $\wedge$, and two temporal operators such as next $\bigcirc$ and until $\mathcal{U}$[2]. Its syntax can be inductively defined as

$$\phi := \text{True} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc\phi \mid \phi_1 \mathcal{U} \phi_2 \,, \tag{7}$$

where $a \in \Pi$ is an atomic proposition. The temporal operators define time-dependent properties when the system evolves. For example, the formula $\bigcirc\phi$ can be read as "ϕ is true at the next state" while $\phi_1 \mathcal{U} \phi_2$ as "ϕ_1 is true at each state until ϕ_2 is true at some future states."

A word is defined as an infinite sequence $\boldsymbol{o} = o_0 o_1 \ldots$ with $o_i \in 2^{\Pi}$. Here denotes $\models$ the satisfaction relation. Consequently, words can be used to interpret the LTL formula's semantics, defined as below.

$$\begin{aligned}
&\boldsymbol{o} \models \text{True} \\
&\boldsymbol{o} \models \alpha && \Leftrightarrow \alpha \in L(\boldsymbol{o}[0]) \\
&\boldsymbol{o} \models \phi_1 \wedge \phi_2 && \Leftrightarrow \boldsymbol{o} \models \phi_1 \text{ and } \boldsymbol{o} \models \phi_2 \\
&\boldsymbol{o} \models \neg\phi && \Leftrightarrow \boldsymbol{o} \mid\neq \phi \\
&\boldsymbol{o} \models \bigcirc\phi && \Leftrightarrow \boldsymbol{o}[1:] \models \phi \\
&\boldsymbol{o} \models \phi_1 \mathcal{U} \phi_2 && \Leftrightarrow \exists t_1 \text{ s.t. } \boldsymbol{o}[t_1:] \models \phi_2, \forall t_2 \in [0, t_1), \boldsymbol{o}[t_2:] \models \phi_1
\end{aligned}$$

In addition to the standard operators introduced in Equation (7), we can derive other propositional and temporal logic operators, including False$\neg$, disjunction $\vee$, implication $\rightarrow$, always $\Box$, and eventually $\Diamond$[2]. It shall be noted that the basic formulas of other temporal operators, $\Box\phi$ and $\Diamond\phi$, can be read as "ϕ is always true in the future" and "ϕ could be true sometimes in the future," respectively. Thus, in an RL problem, an LTL formula can describe whether or not a set of infinite traces (i.e., sequences of MDP states) can satisfy the user-specified task(s).

Example 1 Given a grid-world, as shown in Fig. 2, an MDP $\mathcal{M}$ can be defined with $s = \{s_0, s_1, ..., s_8\}$ and $\Pi = \{T_1, T_2, T_3, U_s\}$. T_i where $i \in 1, 2, 3$ are labeled on the states of interest while U_s is labeled on an unsafe state. Considering a user-specified task, "First T_1, then T_2, finally T_3, and never U_s," we can write an LTL formula as $\varphi = \Diamond(T_1 \wedge \Diamond(\mathrm{T}_2 \wedge \Diamond \mathrm{T}_3)) \wedge \neg U_s$. This task can be satisfied by, for example, a word of $\boldsymbol{o} = T_1 T_1{}^* T_2 T_3 T_3{}^*$ where $*$ matches the preceding character for 0 or more times (up to infinite). Consequently, a set of infinite traces will satisfy the task defined in this example if it can generate the above-expressed word.

In the above example, the solution consists of any policy that can generate traces to satisfy the LTL-formulated task(s) as well as maximize the expected return in Equation (5). In other words, the LTL formula plays a constraint in finding the optimal policy. However, such a constraint cannot be directly implemented in conventional RL problems. Instead, a finite state automaton is usually employed to represent the LTL formula. Then, the optimal policy can be achieved via RL with automaton theory and model checking[2].

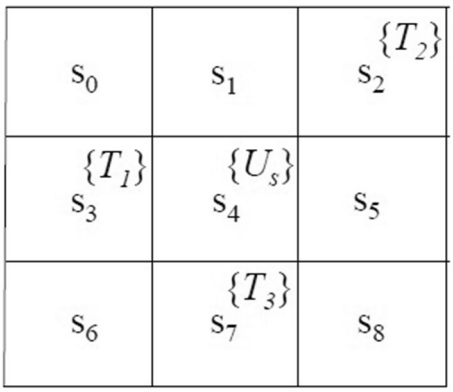

Figure 2. A grid-world with states of interest labeled by T_i and unsafe states labeled by U_s.

Limit-deterministic generalized Büchi automaton. As discussed above, an LTL formula can specify a complex high-level task. Then, task satisfaction can be evaluated by an automaton, such as an LDGBA[18], through model checking.

Definition 1 *(LDGBA)* An LDGBA, $\mathcal{A} = \left(Q, \Sigma, \delta, q_0, F\right)$, consists of a finite set of states Q and a finite alphabet $\Sigma = 2^{\Pi}$ with a set of atomic propositions Π. The transition function, $\delta(q, \alpha) : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^{Q}$, allows the LDGBA to change its state when taking an input symbol ($\alpha \in \Sigma$) or not ($\alpha \in \{\epsilon\}$). In addition, $q_0 \in Q$ is an initial automaton state. $F = \left\{F_1, F_2, \ldots, F_f\right\}$ represents a set of accepting sets in the automaton, and $F_i \subseteq Q$ where $\forall i = 1, \ldots f$. The state set Q in an LDGBA can be divided into deterministic and non-deterministic sets, i.e., Q_D and Q_N, respectively. Such partition satisfies the following:

- The LDGBA transitions in the deterministic set are total, i.e., $|\delta\left(q, \alpha\right)| = 1$ with $\alpha \in \Sigma$. Such transitions are allowed within this set only. Therefore, after consuming an input symbol α at an automaton state q, the resulted automaton state is $\delta\left(q, \alpha\right) \subseteq Q_D$,
- The state transitions without any input symbol, i.e., ϵ-transitions, are only valid for state transitions from the non-deterministic set (Q_N) to the deterministic set (Q_D). Therefore, they are not allowed in Q_D. and
- All the accepting sets are subsets of the deterministic set.

A run of an LDGBA can be written as $\boldsymbol{q} = q_0 q_1 \ldots$. Let $\inf\left(\boldsymbol{q}\right)$ denotes the infinite portion of $\boldsymbol{q}$. If there exists $\inf\left(\boldsymbol{q}\right) \cap F_i \neq \emptyset, \forall i \in \left\{1, \ldots f\right\}$, we say that $\boldsymbol{q}$ satisfies the LDGBA acceptance condition or the LDGBA accepts $\boldsymbol{q}$. Example 2 demonstrates an LDGBA, representing the LTL formula in Example 1. We recommend Owl[36] to readers for more details about automaton generation.

Example 2 In Example 1, the LTL formula is expressed as $\phi = \Diamond(T_1 \wedge \Diamond(\mathtt{T}_2 \wedge \Diamond \mathtt{T}_3)) \wedge \neg U_s$ for the user-specified task. Figure 3 shows the LTL-induced LDGBA, which has only one accepting set $F_1 = \{q_3\}$. A run of LDGBA, for example, $\boldsymbol{q} = q_0 q_1 q_2 q_3$, is accepted by this LDGBA because $\boldsymbol{q} \cap F_1 = \{q_3\}$.

Problem formulation. It was discussed above that an LTL formula ϕ can describe the task specifications to be performed by an agent. For example, given a policy $\boldsymbol{\xi} = \xi_0 \xi_1 \ldots$ that the agent learns from MDP $\mathcal{M}$, it can generate a path, denoted by $\boldsymbol{s}_{\infty}^{\boldsymbol{\xi}} = s_0 \ldots s_i s_{i+1} \ldots$, with $p_S(s_i, a_i, s_{i+1}) > 0$. Correspondingly, a sequence of labels (i.e., a trace) can be derived from the labeling function as $L\left(\boldsymbol{s}_{\infty}^{\boldsymbol{\xi}}\right) = l_0 l_1 \ldots$ where $l_i \in L(s_i)$. If this trace satisfies the task, i.e., $L\left(\boldsymbol{s}_{\infty}^{\boldsymbol{\xi}}\right) \models \phi$, the satisfaction probability that the agent can accomplish the task is expressed as

$$\Pr{}_{\mathcal{M}}^{\boldsymbol{\xi}}(\phi) = \Pr{}_{\mathcal{M}}^{\boldsymbol{\xi}}\left(L\left(\boldsymbol{s}_{\infty}^{\boldsymbol{\xi}}\right) \models \phi \middle| \boldsymbol{s}_{\infty}^{\boldsymbol{\xi}} \in \boldsymbol{S}_{\infty}^{\boldsymbol{\xi}}\right) \tag{8}$$

where $\boldsymbol{S}_{\infty}^{\boldsymbol{\xi}}$ is defined as a set of admissible paths generated from policy $\boldsymbol{\xi}$, starting with the initial state s_0.

Assumption 1 At least one deterministic policy exists such that the agent can accomplish the task with a non-zero probability by following this policy.

Assumption 1 states that the agent can always find a policy from which an induced trace can fully satisfy the user-specified LTL task. This assumption is moderate and has been widely utilized[11,14,37]. This paper considers the LTL task as the form $\phi = \phi_g \wedge \Box \phi_{safe}$, where ϕ_g provides a general form of high-level tasks, and ϕ_{safe} represents the safety requirement. Consequently, we define the problem as follows.

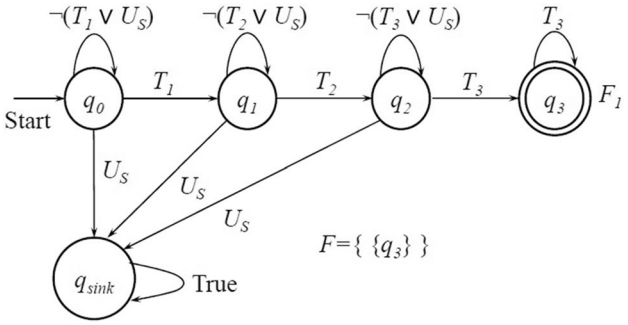

Figure 3. An LDGBA for the LTL formula $\phi = \Diamond(T_1 \wedge \Diamond(\mathtt{T}_2 \wedge \Diamond \mathtt{T}_3)) \wedge \neg U_s$.

Problem 1 This RL problem considers (1) a user-specified task that can be formulated via LTL as $\phi = \phi_g \wedge \Box\phi_{safe}$ and (2) an MDP $\mathcal{M}$ in which the transition function is unknown. The objective is to find a deterministic policy $\boldsymbol{\xi}^*$, which can maximize the probability of task satisfaction, i.e., $\boldsymbol{\xi}^* = \text{argmax}_{\boldsymbol{\xi}} \Pr^{\boldsymbol{\xi}}_{\mathcal{M}}(\phi_g)$, and maintain the safety ϕ_{safe} during the learning process.

To address Problem 1, Section "Automaton-based reward design" proposes an automaton-based reward design to guide the agent in learning the optimal policy on the product of MDP and automaton. Then, Section "Safety value functions" develops a safe padding technique by introducing safety value functions to promote safety during the learning process. In addition, a quantum action selection technique is proposed in Section "Quantum action selection" to substitute the ε-greedy algorithm for the balance of exploration and exploitation in learning.

Automaton-based reward design

Previous work[15] has shown that directly utilizing LDGBA may fail to find deterministic policies satisfying LTL specifications. This issue can be addressed by designing an E-LDGBA.

Embedded LDGBA. To keep tracking unvisited accepting sets in an LDGBA, we introduce a tracking-frontier set T during model checking. In addition, a Boolean variable, $\mathcal{B}$, is employed to indicate the satisfaction of accepting conditions for each round. Here, one round is defined as all accepting sets being visited. On the other hand, this tracking-frontier set is initialized as $T = F$ while $\mathcal{B}$ as False. Then, the set $(T, \mathcal{B}) = f_V(q, T)$ is updated synchronously during learning process as:

$$f_V(q,T) = \begin{cases} (T \setminus F_j, \text{False}), & \text{if } q \in F_j \text{ and } F_j \in T, \\ (F \setminus F_j, \text{True}) & \text{if } q \in F_j \text{ and } T = \emptyset, \\ (T, \text{False}), & \text{otherwise.} \end{cases} \tag{9}$$

Definition 2 *(E-LDGBA)* Considering an LDGBA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and a Boolean variable $\mathcal{B}$, we can define an E-LDGBA correspondingly as $\overline{\mathcal{A}} = (\overline{Q}, \Sigma, \overline{\delta}, \overline{q}_0, \overline{F}, f_V, T, \mathcal{B})$ with initially setting $T = F$ and $\mathcal{B}$ =False, respectively. The automaton states are augmented, i.e., $\overline{Q} = Q \times 2^F$ or $\overline{q} = (q, T)$. In the transition function, $\overline{\delta} : \overline{Q} \times (\Sigma \cup \{\epsilon\}) \to 2^{\overline{Q}}$, the finite alphabet Σ is the same as in LDGBA. In addition, the non-deterministic set $\overline{Q}_N$, the deterministic set $\overline{Q}_D$, and ϵ-transitions from $\overline{Q}_N$ to $\overline{Q}_D$ can be correspondingly constructed. Similarly, the set of accepting sets in E-LDGBA is $\overline{F} = \{\overline{F}_1, \overline{F}_2 \ldots \overline{F}_f\}$, where $\overline{F}_j = \{(q, T) \in \overline{Q} | q \in F_j \wedge F_j \subseteq T\}, j = 1, \ldots f$. In particular, the transition $\overline{q}' = \overline{\delta}(\overline{q}, \overline{\sigma})$, where $\overline{\sigma} \in (\Sigma \cup \{\epsilon\})$, shall satisfy the following: 1) the transitions in LDGBA are valid, i.e., $q' = \delta(q, \overline{\sigma})$, and 2) T and $\mathcal{B}$ are synchronously updated via $(T, \mathcal{B}) = f_V(q', T)$ at each transition as defined in Eq. (9).

We abuse the tuple structure in Definition 2 because the tacking-frontier set T and the Boolean variable $\mathcal{B}$ are synchronously updated after each transition. While updating the frontier set T, the Boolean variable $\mathcal{B}$ indicates whether the current state belongs to at least one accepting set that has not been visited within the current round. This design is essential to guide the agent to visit all accepting sets once per round with infinite rounds based on user-specified tasks.

In the rest of this paper, given an LTL formula ϕ, we utilize $\mathcal{A}_\phi$ and $\overline{\mathcal{A}}_\phi$ to represent an LDGBA and its corresponding E-LDGBA, respectively. Here assume that $\mathcal{L}(\mathcal{A}_\phi) \subseteq \Sigma^\omega$ is the language accepted by $\mathcal{A}_\phi$. In other words, this language is the set of all infinite words that satisfy the LTL formula ϕ[2]. If $\mathcal{L}(\overline{\mathcal{A}}_\phi) \subseteq \Sigma^\omega$ is the language accepted by $\overline{\mathcal{A}}_\phi$, we have the following lemma.

Lemma 1 *Given an LTL formula ϕ, we can generate two automata: an LDGBA $\mathcal{A}_\phi = (Q, \Sigma, \delta, q_0, F)$ and its corresponding E-LDGBA $\overline{\mathcal{A}}_\phi = (\overline{Q}, \Sigma, \overline{\delta}, \overline{q}_0, \overline{F}, f_V, T, \mathcal{B})$. Then, it leads that*

$$\mathcal{L}(\overline{\mathcal{A}}_\phi) = \mathcal{L}(\mathcal{A}_\phi). \tag{10}$$

The proof of Lemma 1 is in our previous work[15]. Therefore, we can employ E-LDGBA to uphold task satisfaction. In addition, we define a set of sink components in an E-LDGBA as below.

Definition 3 A set of non-accepting sink components in an E-LDGBA, i.e., $\overline{Q}_{sink} \subseteq \overline{Q}$, is a group of automaton states from which none of the accepting states can be reached.

In addition, after defining the LTL-formulated task as $\phi = \phi_g \wedge \Box\phi_{safe}$, we can always find a set of states $\overline{Q}_{unsafe} \subseteq \overline{Q}_{sink}$ associated with the violation of ϕ_{safe}.

Embedded product MDP (EP-MDP).

Definition 4 *(EP-MDP)* Considering an MDP $\mathcal{M} = (S, A, p_S, s_0, \Pi, L)$ and an LTL-converted E-LDGBA $\overline{\mathcal{A}}_\phi = (\overline{Q}, \Sigma, \overline{\delta}, \overline{q}_0, \overline{F}, f_V, T, \mathcal{B})$, we can construct an embedded product MDP (EP-MDP), $\mathcal{P} = \mathcal{M} \times \overline{\mathcal{A}}_\phi = (X, U^{\mathcal{P}}, p^{\mathcal{P}}, x_0, F^{\mathcal{P}}, f_V, T, \mathcal{B})$, which consists of a set of labeled states $X = S \times 2^\Pi \times \overline{Q}$ with

$x = (s, l, \overline{q}) = (s, l, q, T) \in X$ and $l \in L(s)$ and a set of actions $U^{\mathcal{P}} = A \cup \{\epsilon\}$. Corresponding to the restriction in the E-LDGBA, ϵ-transitions are valid from $x = (s, l, \overline{q})$ with $\overline{q} \in \overline{Q}_N$ to $x' = (s, l, \overline{q}')$ with $\overline{q}' \in \overline{Q}_D$ only. The transition probability $p^{\mathcal{P}}(x, u^{\mathcal{P}}, x') : X \times U^{\mathcal{P}} \times X \to [0, 1]$ equals (1) $p_S(s, a, s')$ if $\overline{\delta}(\overline{q}, l) = \overline{q}'$ and $u^{\mathcal{P}} = a \in A(s)$; (2) 1 if $u^{\mathcal{P}} \in \{\epsilon\}, \overline{q}' \in \overline{\delta}(\overline{q}, \epsilon)$, and $(s', l') = (s, l)$; and (3) 0 otherwise. In addition, $x_0 = (s_0, l_0, \overline{q}_0)$ is the initial state, and $F^{\mathcal{P}} = \left\{F_1^{\mathcal{P}}, F_2^{\mathcal{P}} \dots F_f^{\mathcal{P}}\right\}$ is the set of accepting sets where $F_j^{\mathcal{P}} = \left\{(s, l, \overline{q}) \in X \middle| \overline{q} \in \overline{F}_j\right\}$, $j = 1, \dots f$. After each transition is completed, T and $\mathcal{B}$ are synchronously updated via $(T, \mathcal{B}) = f_V(q', T)$ in Equation (9).

It shall be noted that the EP-MDP $\mathcal{P}$ can capture the dynamical interchanges between all possible paths over the associated MDP and all words recognized by the corresponding E-LDGBA. Assuming $\boldsymbol{\pi}$ is a policy over $\mathcal{P}$ and generates an infinite path, e.g., $\boldsymbol{x}_{\infty}^{\boldsymbol{\pi}} = x_0 \dots x_i x_{i+1} \dots$. This path is acceptable if $\inf\left(\boldsymbol{x}_{\infty}^{\boldsymbol{\pi}}\right) \cap F_i^{\mathcal{P}} \neq \emptyset$ where $i = 1, \dots f$. Such an accepting path $\boldsymbol{x}_{\infty}^{\pi}$ can produce a policy $\boldsymbol{\xi}$ over $\mathcal{M}$ to satisfy the LTL-specified task ϕ. On the other hand, $\mathcal{P}$ has at least one accepting maximum end component (AMEC)[2], $\mathcal{P}'_{(X', U')}$, and reaching this AMEC is the same as satisfying task ϕ.

As mentioned above, this study decomposes an LTL-formulated task into $\phi = \phi_g \wedge \Box\phi_{safe}$, and we can define a set of unsafe states as $X_{unsafe} = \left\{x = (s, l, \overline{q}) \in X | s \in S \wedge \overline{q} \in \overline{Q}_{unsafe}\right\}$. Also, let $\Pr^{\boldsymbol{\pi}}[x \models \mathrm{Acc}_{\mathcal{P}}]$ denote the probability of policy $\boldsymbol{\pi}$ satisfying the EP-MDP's acceptance conditions. Then, the maximum probability of satisfying the acceptance of $\mathcal{P}$ is defined as $\Pr_{max}[x \models \mathrm{Acc}_{\mathcal{P}}] = \max_{\boldsymbol{\pi}} \Pr_{\mathcal{M}}^{\boldsymbol{\pi}}(\mathrm{Acc}_{\mathcal{P}})$. Consequently, Problem 1 can be rephrased as below.

Problem 2 Considering an MDP $\mathcal{M}$ and an LTL task ϕ, a corresponding EP-MDP $\mathcal{P}$ exists, and transition probabilities are unknown. The objective is to find a policy $\boldsymbol{\pi}^*$ that satisfies the EP-MDP's acceptance conditions with a maximum probability, i.e., $\Pr^{\boldsymbol{\pi}^*}[x \models \mathrm{Acc}_{\mathcal{P}}] = \Pr_{max}[x \models \mathrm{Acc}_{\mathcal{P}}]$, and avoid entering X_{unsafe} during the learning process.

A base reward design will be discussed in the following subsection. Such a design can enable the RL agent to find the optimal policy and achieve the maximum probability of satisfying task(s). Then, we will further improve the reward density via reward shaping with a potential function.

Base reward. In an EP-MDP $\mathcal{P}$, all accepting states can be grouped into a set as $F_U^{\mathcal{P}} = \left\{x \in X | x \in F_i^{\mathcal{P}}, \forall i \in \{1, \dots f\}\right\}$. According to the reward function $\Lambda(s, a, s')$ defined in an MDP $\mathcal{M}$, after each transition $(x, u^{\mathcal{P}}, x')$ in the corresponding EP-MDP, the agent can receive a reward as: 1) $R(x, u^{\mathcal{P}}, x') = \Lambda(s, a, s')$ if $\overline{\delta}(\overline{q}, l) = \overline{q}'$ and $u^{\mathcal{P}} = a \in A(s)$; and 2) $R(x, u^{\mathcal{P}}, x') = 0$ otherwise. In addition, the discount factor in the EP-MDP can be defined similarly. In this study, we consider both functions depending on the state only. Consequently, inspired by[14], the reward and discount factor functions can be designed as below.

$$R(x) = \begin{cases} 1 - r_F, & \text{if } x \in F_U^{\mathcal{P}}, \\ 0, & \text{otherwise}, \end{cases} \tag{11}$$

$$\gamma(x) = \begin{cases} r_F, & \text{if } x \in F_U^{\mathcal{P}}, \\ \gamma_F, & \text{otherwise}, \end{cases} \tag{12}$$

where γ_F is the discount factor when the agent doesn't reach any accepting state after a valid transition, and $r_F(\gamma_F)$ otherwise. In addition, $r_F(\gamma_F)$ shall satisfy $\lim_{\gamma_F \to 1^-} r_F(\gamma_F) = 1$ and $\lim_{\gamma_F \to 1^-} \frac{1-\gamma_F}{1-r_F(\gamma_F)} = 0$. Based on the approval in[14], a state value represents the probability that an agent can accomplish the specified task starting from this state.

Reward shaping. The reward function designed above is always zero for any transition between the states $x \notin F_U^{\mathcal{P}}$. For instance, given an LDGBA $\mathcal{A}_\phi$ as shown in Fig. 3, we can obtain its corresponding E-LDGBA $\overline{\mathcal{A}}_\varphi$ and construct the EP-MDP $\mathcal{P} = \mathcal{M} \times \overline{\mathcal{A}}_\phi$. Then, it can be observed that executing any transition between product states $x = (s, l, q_1, T)$ and $x' = (s', l', q_2, T')$ always renders a zero reward for any MDP state $s, s' \in S$. We propose a potential function $\Phi(x) : X \to \mathbb{R}$ to increase the reward density and redesign the reward function as below.

$$R'\left(x, u^{\mathcal{P}}, x'\right) = R(x) + \gamma(x) \cdot \Phi\left(x'\right) - \Phi(x) \tag{13}$$

Given an EP-MDP $\mathcal{P} = \mathcal{M} \times \overline{\mathcal{A}}_\phi = \left(X, U^{\mathcal{P}}, p^{\mathcal{P}}, x_0, F^{\mathcal{P}}, f_V, T, \mathcal{B}\right)$, there exists a set automaton accepting states, i.e., $\overline{Q}_F = \left\{\overline{q} \in \overline{Q} | \overline{q} \in \overline{F}_i, \forall i \in \{1, \dots f\}\right\}$. If an agent visits a product state $x = (s, l, q_1, T) = (s, l, \overline{q})$ and it is its first time to visit the associated automaton state $\overline{q}$ that belongs to $\overline{Q} \setminus \left(\overline{Q}_F \cup \overline{q}_0 \cup \overline{Q}_{sink}\right)$, the agent will receive a positive reward. Such a modification on the reward design will enhance the guiding of task satisfaction during the learning process via model checking because any automaton state in $\overline{Q} \setminus \left(\overline{Q}_F \cup \overline{q}_0 \cup \overline{Q}_{sink}\right)$, i.e., the one that can reach any accepting set, has to be explored starting from the initial automaton state $\overline{q}_0$.

We design another tracking-frontier set T_Φ to keep tracking unvisited automaton states in $T_{\Phi 0} = \overline{Q} \setminus \left(\overline{Q}_F \cup \overline{q}_0 \cup \overline{Q}_{sink}\right)$. It shall be noted that T_Φ is different from T, defined in Section "Embedded

LDGBA", which tracks unvisited accepting sets. Initially, T_Φ is set as $T_{\Phi 0}$. Then, it is updated as below at each transition from $(s, l, \overline{q})$ to $(s', l', \overline{q}')$ after taking action $u^{\mathcal{P}}$.

$$f_\Phi(\overline{q}', T_\Phi) = \begin{cases} T_\Phi \setminus \overline{q}', & \text{if } \overline{q} \in T_\Phi, \\ T_{\Phi 0} \setminus \overline{q}' & \text{if } \mathcal{B} = \text{True}, \\ T_\Phi, & \text{otherwise.} \end{cases} \tag{14}$$

After $\mathcal{B}$ becomes True in f_V , i.e., the agent has visited all accepting sets, T_Φ will be reset as $T_{\Phi 0}$. Consequently, the potential function $\Phi(x)$ at $x = (s, l, \overline{q})$ is proposed as:

$$\Phi(x) = \begin{cases} 1 - r_F, & \text{if } \overline{q} \in T_\Phi, \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

where r_F is the discount factor as defined in Section "Base reward". According to Equation (15), the potential function equals $1 - r_F$ or 0 for unvisited or visited automaton states. This design will enhance the efficiency of exploration. Based on the shaped reward in Equation (13), the expected return that an agent can collect from state $x \in X$ under policy $\boldsymbol{\pi}$ can be expressed as

$$U^{\boldsymbol{\pi}}(x) = \mathbb{E}^{\boldsymbol{\pi}}\left[\sum_{i=0}^{\infty} \gamma^i(x_i) \cdot R'\left(x_i, u^{\mathcal{P}}, x_{i+1}\right) | x_0 = x\right]. \tag{16}$$

Theorem 1 *Given an EP-MDP* $\mathcal{P} = \mathcal{M} \times \overline{\mathcal{A}}_\phi$, *by selecting* $\gamma_F \to 1^-$ *and applying a shaped reward function* (13), *the optimal policy* $\boldsymbol{\pi}^*$ *can maximize the expected return in Equation* (16). *This policy also maximizes the probability of task satisfaction, i.e.,* $\Pr^{\boldsymbol{\pi}^*}[x \models \text{Acc}_{\mathcal{P}}] = \Pr_{max}[x \models \text{Acc}_{\mathcal{P}}]$.

Proof First, theorems 1-3 of[15] have verified that by applying the base reward design in Equations (11) and (12), optimizing the expected return can guarantee the optimal policy satisfying the specified LTL task with the maximum probability. Then, the work of[38] has shown that such an optimal policy remains invariant by applying a shaped reward in Eq. (13). □

To be brief, we proposed a reward-shaping scheme to overcome the issues of sparse reward in an EP-MDP and guarantee that an agent can learn the optimal policy to maximize the probability of task satisfaction.

Safety value functions

State and action safety values. Given an MDP $\mathcal{M} = (S, A, p_S, s_0, \Pi, L)$, the agent is located at the current state $s \in S$. Assuming that the agent can observe the label of its current state only but can record the safety status of the states it has visited as

$$u_s(s) = \begin{cases} 0 & \text{if } L(s) \subseteq L_{us} \\ 1 & \text{otherwise} \end{cases} \tag{17}$$

where $L_{us} \subset \Pi$ is a set of unsafe labels, which is a subset of the set of atomic propositions, Π.

The transition probability p_S is assumed to be unknown to the agent. However, the agent can estimate the transition dynamics based on the observation history. In other words, the agent also records the number of times it executes action a at state s, i.e., $\mathcal{N}(s, a)$, and the number of times it reaches the next state s' after taking a at s, i.e., $N(s, a, s')$. Consequently, the current belief of the agent about its transition dynamics can be expressed below by considering the Maximum Likelihood Estimation (MLE)[39] for the mean transition probability function.

$$\tilde{p}_S(s, a, s') = \frac{N(s, a, s')}{\mathcal{N}(s, a)} \tag{18}$$

A state safety value function $V_s(s)$ is introduced here to represent the minimum probability of the agent moving to a safe state after taking action. It can be estimated via the Bellman update[17,40]:

$$V_s(s) = \min_{a \in A(s)} \sum_{s'} \tilde{p}_S(s, a, s') u_s(s'). \tag{19}$$

Then, here defines an action safety value function, representing the maximum probability of the agent staying safe after taking action a at state s, as

$$Q_s(s, a) = \sum_{s'} \tilde{p}_S(s, a, s') V_s(s'). \tag{20}$$

If considering environment uncertainty, a labeling probability function exists such that a state may be labeled "unsafe" with a probability. Consequently, Equation (17) needs to be rewritten as below to update the state's safety status at every visit.

$$u_s(s) = \begin{cases} u_s(s) - \frac{u_s(s)}{N_s(s)+1} & \text{if } L(s) \subseteq L_{us} \\ u_s(s) + \frac{1.0-u_s(s)}{N_s(s)+1} & \text{otherwise} \end{cases} \tag{21}$$

where $N_s(s)$ is the total number of times the agent has visited state s, and it needs to be updated afterward as $N_s(s) \leftarrow N_s(s) + 1$.

In addition, it shall be noted that the safety value functions can be calculated in a continuous state space by revising Equations (19) and (20) as

$$V_s(s) = \min_{a \in A(s)} \frac{1}{\Omega} \int_{\Omega} \overline{P}_S(s, a, s') u_s(s') ds' \tag{22}$$

and

$$Q_s(s, a) = \frac{1}{\Omega} \int_{\Omega} \overline{P}_S(s, a, s') V_s(s') ds' \tag{23}$$

where $\overline{P}_S(s, a, s')$ can be predicted by an artificial neural network in addition to Q-networks if deep Q-learning[41] is used. Similar to Q-networks, this transition probability network can be trained and updated by a collection of experiences. In this paper, we consider discrete state spaces only.

Safe reinforcement learning. Similar to Equation (6), in a Q-learning[33] on an EP-MDP $\mathcal{P} = \left(X, U^{\mathcal{P}}, p^{\mathcal{P}}, x_0, F^{\mathcal{P}}, f_V, T, \mathcal{B}\right)$, Q values can be updated below after the agent takes action $u^{\mathcal{P}}$ and moves from one state (x) to another state (x').

$$\begin{aligned} Q\left(x, u^{\mathcal{P}}\right) \leftarrow & (1-\alpha) Q\left(x, u^{\mathcal{P}}\right) \\ & + \alpha \left[R'\left(x, u^{\mathcal{P}}, x'\right) + \gamma(x) \cdot \max_{\overline{u}^{\mathcal{P}} \in U^{\mathcal{P}}} Q\left(x', \overline{u}^{\mathcal{P}}\right) \right] \end{aligned} \tag{24}$$

The action safety value function in (20) can be incorporated with the Q-value for safe learning. It shall be noted that ϵ-transitions don't influence the generated policy. Therefore, the action selection technique during the learning process in an EP-MDP $\mathcal{P}$ is proposed as:

- if $\{\epsilon\} \subset U^{\mathcal{P}}(x)$ where $x = (s, l, q, T)$, select action $u^{\mathcal{P}} \in \{\epsilon\}$.
- if $U^{\mathcal{P}}(x) = A(s)$, section action $u^{\mathcal{P}}(x) \in A(s)$ based on the ε-greedy algorithm as

$$u^{\mathcal{P}}(x) = \begin{cases} \underset{a \in A(s)}{\text{argmax}}[Q(x, a) + \beta Q_s(s, a)] & \text{with probability 1-}\varepsilon \\ \text{any action}(a \in A(s)) & \text{with probability } \varepsilon \end{cases} \tag{25}$$

where $\beta \in [0, 1]$ is a parameter bias in selecting safe actions with the importance of action safety. According to the reward function employed in Equation (11), an action value, i.e., $Q(s, a)$, represents the probability of the agent accomplishing the specified task by taking action a at state s. Since both represent probabilities, it is natural to combine the action value function and the action safety value function linearly for decision-making in Equation (25). For other definitions of reward functions, Equation (25) might need to be revised[17].

The safe RL algorithm is described in Algorithm 1, and the product states are generated on the fly based on Definition 4. It shall be noted that the action safety values in Equation (20) can be globally evaluated at the beginning of each learning episode (using collecting data as prior knowledge) as shown in Algorithm 1 or locally updated at the current state and its neighboring states at each step. Our simulations showed that either approach could dramatically reduce the times the agent encounters unsafe states.

Algorithm 1 Safe reinforcement Learning

procedure INPUT: $(\mathcal{M}\ ,\ \phi)$
Output: optimal policy $\boldsymbol{\pi}^*$
Initialization: $episode = 0$, $iteration = 0$ and τ (the maximum allowable learning steps)
set $r_F = 0.99$ and $\gamma_F = 0.9999$ to determine $R(x)$, $\gamma(x)$, and $\Phi(x)$
for all $s \in S$ **do**
$u_s(x) = V_s(s) = 0$ and $Q_s(s,a) = 0$, $\forall a \in A(s)$
$N(s,a,s') = 0$ and $\mathcal{N}(s,a) = 0$, $\forall a \in A(s)$ and $\forall s' \in S$
end for
for all $x \in X$ **do**
$Q\left(x, u^{\mathcal{P}}\right) = 0$, $\forall u^{\mathcal{P}} \in U^{\mathcal{P}}(x)$
$Count\left(x, u^{\mathcal{P}}\right) = 0$, $\forall u^{\mathcal{P}} \in U^{\mathcal{P}}(x)$
end for
$x = x_0$;
while $\boldsymbol{U}$, i.e., the expected return, is not converged **do**
$episode++$;
evaluate action safety values $Q_s(s,a)$ via (20)
$\varepsilon = 1/episode$;
while $iteration < \tau$ **do**
$iteration++$
if $\{\epsilon\} \subset U^{\mathcal{P}}$ **then** Select action $a \in \{\epsilon\}$
else Select action a based on the ε-greedy technique in (25)
end if
Execute $u^{\mathcal{P}} = a$ and observe x'
$Count\left(x, u^{\mathcal{P}}\right)++$
$\alpha = 1/Count\left(x, u^{\mathcal{P}}\right)$
Update $u_s(s)$, $\mathcal{N}(s,a)$, and $N(s,a,s')$
Update $Q\left(x, u^{\mathcal{P}}\right)$ via (24)
Update $U(x)$ via (16)
$x = x'$
end while
end while
for all $x \in X$ **do**
$U(x) = \max_{u^{\mathcal{P}} \in U^{\mathcal{P}}} Q\left(x, u^{\mathcal{P}}\right)$
$\boldsymbol{\pi}^*(x) = \arg\max_{u^{\mathcal{P}} \in U^{\mathcal{P}}} U(x)$
end for
end procedure

Quantum action selection

Grover's algorithm. Grover's algorithm[42], sometimes referred to as Grover Search, quantum search, or quantum database search, is a quantum algorithm for searching through a non-ordered list. It can be used to invert a function, to which there are many possible solutions, but only one is correct. Grover's algorithm can find the solution out of the pack much faster than any classical algorithm. Searching through the database with N items using a classical approach requires $O(N)$ evaluations (on average $N/2$ evaluations). However, Grover's algorithm can complete such a search using only $O(\sqrt{N})$ evaluations.

The search problem we consider here consists of three steps: marking the desired basic state within the 2^n space represented by an n-qubit quantum state, applying Grover's algorithm on the quantum state, and finding the desired basic state with a single measurement. At first, we prepare an n-qubit quantum state, which is initialized as $|q_1 q_2 \ldots q_n\rangle = |00\ldots0\rangle$, by applying an H gate to each qubit,

$$|\psi\rangle = H^n|00\ldots0\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle \tag{26}$$

which results in an equal superposition of all 2^n basic states.

We use an 'oracle' operator U_d to mark the desired basic state, e.g., $|d\rangle$, in the above superposition $|\psi\rangle$, as described in[43]. With the assistance of ancilla qubits, this operator utilizes X gates and an nth-order $CNOT$ gate to flip the sign of the desired state. It shall be noted that the nth-order $CNOT$ gate uses n qubits as the control qubits with the ancilla qubit as the target. Consequently, the n-qubit quantum state becomes

$$|\phi\rangle = U_d|\psi\rangle = \frac{1}{\sqrt{2^n}}\sum_{k\neq d}|k\rangle - \frac{1}{\sqrt{2^n}}|d\rangle \tag{27}$$

Next, a Grover diffusion operation is conducted to amplify the amplitude of the desired basic state. Mathematically, the Grover diffusion operation is equivalent to a reflection of the average amplitude of all 2^n basic states, and it can be expressed as the Grover diffusion operator[43], $U_s = 2|\psi\rangle\langle\psi| - I$, on the n-qubit system, $|\phi\rangle$ in Equation (27). However, $|\psi\rangle\langle\psi|$ is not a unitary operator, so it is not physically realizable. It has been demonstrated that H gates and the oracle operator can be utilized to realize the Grover diffusion operator[43]. Theoretically, the optimal number of times to conduct the Grover iteration, consisting of oracle and diffusion operations, is $r = \frac{\pi}{4\theta} - \frac{1}{2}$, where $\sin\theta = \frac{1}{\sqrt{N}}$ and $N = 2^n$. When $N >> 1$, it can be approximated as $\frac{\pi\sqrt{N}}{4}$.

The last step is to measure the resulting n-qubit quantum state and find the desired basic state. For a two-qubit system, after one Grover iteration, the quantum state becomes

$$U_s U_d H^2|00\rangle = -|d\rangle \tag{28}$$

Theoretically, the desired basic state $|d\rangle$ is surely found. For a three-qubit system, after two Grover iterations, the quantum state becomes

$$(U_s U_d)^2 H^3|000\rangle = -0.08839\sum_{k\neq d}|k\rangle + 0.97227|d\rangle \tag{29}$$

Therefore, there is a 5.5% probability of obtaining a wrong basic state other than $|d\rangle$.

Quantum action selection. Here we propose the quantum action selection technique that can replace the ε-greedy action selection from the quantum computing point of view. According to the number of available actions, e.g., M, n qubits are utilized to form a superposition of 2^n basic states, representing discrete actions in an MDP problem. M and n are characterized by the following inequality:

$$M \leq 2^n < 2M \tag{30}$$

Taking the example of a mobile robot moving in a grid world, if the robot can move "up", "down", "left" and "right", those actions are represented by two-qubit basic states, $|00\rangle$, $|10\rangle$, $|01\rangle$, and $|11\rangle$, respectively. After applying the H gate on each qubit of the initial quantum state $|00\rangle$ as in Equation (26), we can obtain an equal superposition of $\frac{1}{2}(|00\rangle + |10\rangle + |01\rangle + |11\rangle)$, meaning that each action will have the same probability of being selected if we measure this two-qubit state once. To select the best action, the Grover algorithm needs to be applied.

Based on the combination of action values and action safety values, as shown in Equation (25), the basic state corresponding to the action associated with the highest value is marked by the oracle function in Equation (27). Then, the amplitude of this basic state is amplified via the Grover diffusion operation, so the probability of selecting the best action is higher than the others when measuring the quantum state. However, as demonstrated in the previous subsection, after one Grover's iteration on the equal superposition state $\frac{1}{2}(|00\rangle + |10\rangle + |01\rangle + |11\rangle)$, it is reduced to the marked basic state with an amplitude of -1. In other words, only the best action is selected when measuring this quantum state, as shown in Equation (28). This is equivalent to the greedy action selection.

In another case, a three-qubit system is needed if the robot can conduct eight actions. After applying the quantum action selection technique described above, the marked action always has a high probability, 94.5%, of being selected. Consequently, the balance of exploration and exploitation is not achieved, especially at the beginning of learning.

It shall be noted that the above discussions are theoretical and only validated on the quantum simulators on classical computers. Indeed, quantum computers always have decoherence and noise issues, so it would be impossible to follow the exact probabilities when measuring the quantum state. Continuing the above example of a 2-qubit system, if the best action is "up", the basic state $|00\rangle$ is marked. After one Grover iteration, the resulting quantum action state is $-|00\rangle$, and it is then measured with 1024 shots. We executed the simulation in three different ways: 1) using a quantum simulator of qiskit[44], an open-source software development kit (SDK), on a classical computer, 2) using ibmq_Belem (an IBM 5-qubit quantum machine) as the backend on a classical computer, and 3) using ibmq_Belem through the IBM quantum composer online. The calculated probabilities are shown in Fig. 4.

It can be seen that the quantum simulator (on a classical computer) doesn't suffer from different error sources because those errors can be corrected with a small amount of extra storage and logic in the classical computer. Larger errors are induced when using a quantum machine as the backend than when directly using a quantum machine. Although some quantum error correction algorithms exist, they utilize many qubits, and only a few qubits are left for actual quantum computation. Therefore, due to the limited number of qubits on the existing quantum computers, no hardware platform can currently conduct robust error corrections for large-scale quantum computation. As mentioned in[45], quantum technologists are putting their efforts into more accurate quantum gates and, eventually, fully error-tolerant quantum computing.

However, in our quantum action selection method, we take advantage of gate noises and measurement errors to balance the exploration and exploitation when using the quantum simulator on a classical computer.

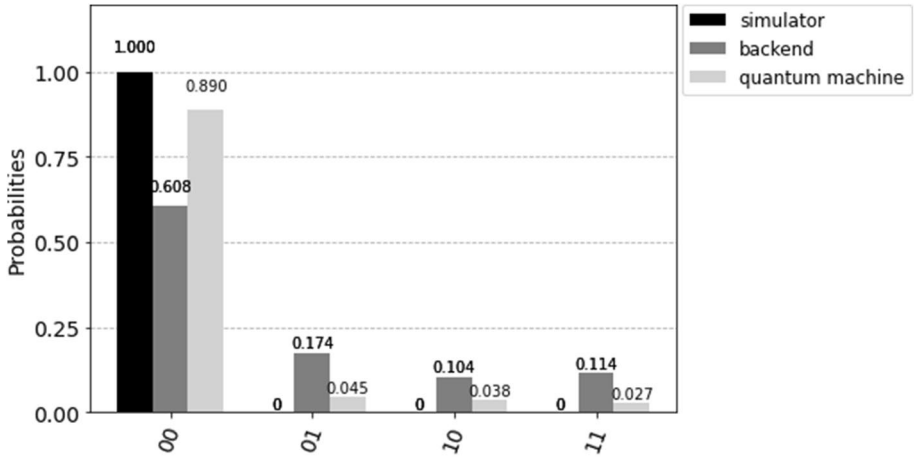

Figure 4. The measurement probability of $-1|00\rangle$ by using 1) a quantum simulator on a classical computer, 2) a quantum machine as the backend on a classical computer, and 3) a quantum machine.

We employ the noise model from qiskit to implement the depolarizing and measurement noise. The first noise model results in an imperfection in quantum gate operations, i.e., replacing the state of any qubit with a completely random state with a probability p_{gate}. Our quantum action selection method applies this noise model on H, X, and $CNOT$ gates. The other noise model flips between $|0\rangle$ and $|1\rangle$ immediately before measurement with probability p_{meas}. It shall be noted that p_{gate} and p_{meas} are heuristic parameters, and it is recommended that the values are initialized as $p_{gate} = p_{meas} = 0.1$ and then reduced to 0.01 for 2-qubit quantum states in our simulations. The probabilities of action selection, corresponding to the example in Fig. 4 at different noise levels, are shown in Fig. 5.

The flowchart of the quantum action selection method is summarized below. We implement the developed quantum action selection technique in our safe RL method (Algorithm 1), resulting in a Quantum Safe Q-learning (QSQ-learning) method. Algorithm 2 can be implemented in any other RL algorithm to replace ε -greedy for action selection.

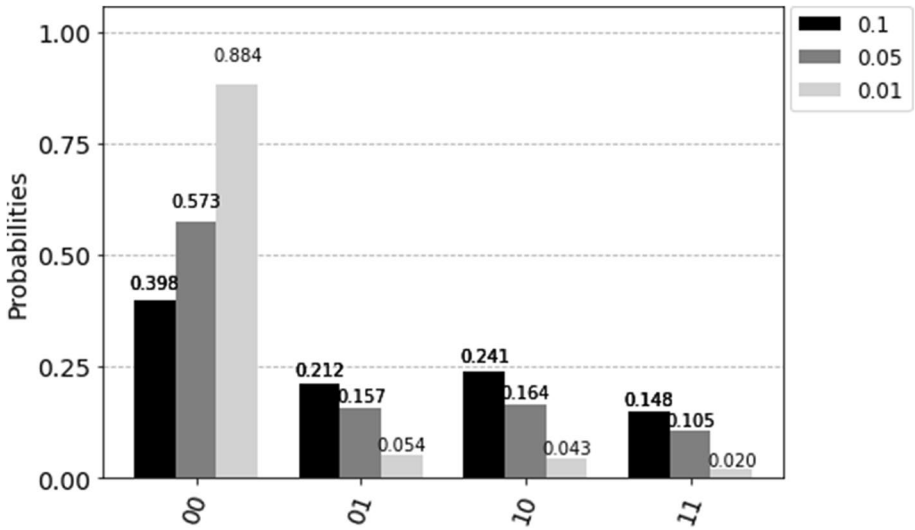

Figure 5. The measurement probability with three different noise levels.

Algorithm 2 Quantum Action Selection

procedure INPUT: (action values, action safety values, the number of actions M)
  Output: action
  Initialization: set $|q_1 q_2 \dots q_n\rangle = |00\dots0\rangle$ where $M \leq 2^n < 2M$
  apply H gate on each qubit to obtain an equal superposition state
  implement the noise models to quantum gates and measurement operations
  while $iteration < \frac{\pi}{4\theta} - \frac{1}{2}$ where $\sin\theta = \frac{1}{\sqrt{N}}$ **do**
    $iteration++$;
    apply the oracle operator to mark the basic state corresponding to the best action
    conduct the Grover diffusion operation
  end while
  measure the quantum state with one shot
end procedure

Simulations and discussions

We study two examples to apply the developed QSQ-learning algorithm for motion planning in grid worlds in this study. The noise in the quantum simulator is set as 0.1 and then slowly decreased to 0.01 for action selection. The reward calculation uses $\gamma_B = 0.99$ and $\gamma = 0.99999$ in Eq. (11).

Motion planning with safe absorbing states. Figure 6 shows a grid world with safe absorbing states[13]. A mobile robot can take four actions: *left*, *up*, *right*, and *down*. The robot receives a reward if it reaches states labeled *a* or *b*. Also, three states, with circles, are safe absorbing states. Once the robot reaches one of them, it will stay in this state no matter which action is taken. States labeled *c* are unsafe, and "Obs" represents an obstacle. In this example, the objective of motion planning is finding a policy so that the robot can eventually always visit a safe absorbing state while avoiding unsafe states. This task can be specified as the below LTL formula.

$$\varphi_1 = (\Diamond\Box a \vee \Diamond\Box b) \wedge \Box\neg c \tag{31}$$

It shall be noted that state (1, 2) (labeled *a*) is not a safe absorbing state. Therefore, although the robot receives a reward for visiting this state, it moves away after taking any action. We first consider deterministic actions, i.e., there is only a single next state when the robot takes action at the current state. We conducted five simulations with safety value functions and another five without safety value functions and obtained the same optimal policies. Figure 7 illustrates state values and an optimal policy. It can be seen that all states, except unsafe and obstacle states, ensure the 100% probability of satisfying the task if the robot starts from one of those states and follows the optimal policy. It is worth mentioning that there is more than one optimal policy for motion planning in this example.

We conduct the simulations with and without safety value functions for comparison. Each simulation takes 10,000 episodes with 200 steps per episode. We investigate the number of times the robot visits unsafe states

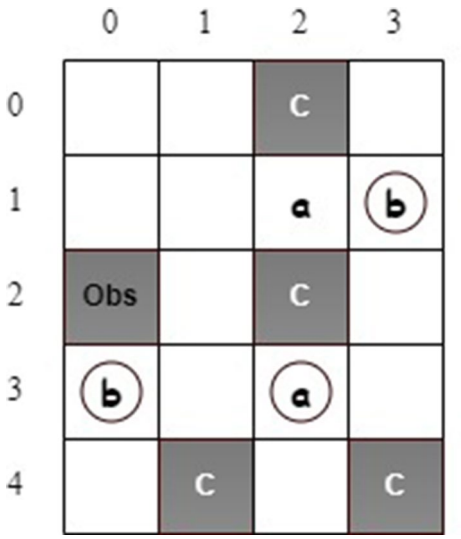

Figure 6. The grid-world with safe absorbing states.

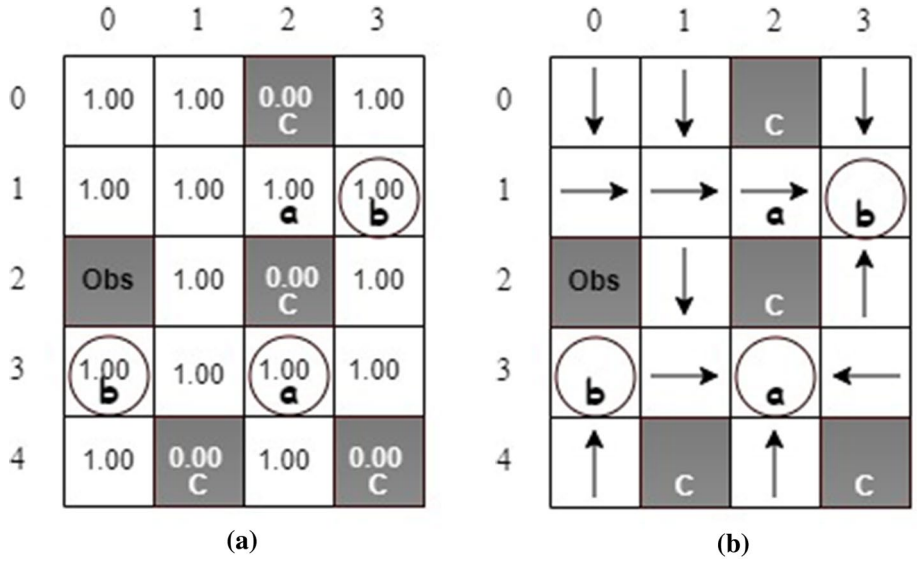

Figure 7. The schemes of (**a**) state values and (**b**) an optimal policy.

while learning optimal policies. It is found that implementing safety value functions could dramatically reduce the number of times to visit unsafe states. Specifically, without using safety value functions, the robot visits unsafe states an average of 950 times, while it visits unsafe states only 177 times when considering both safety values and Q-values in decision-making.

The state safety values and the maximum action safety values are shown in Fig. 8. The safety state value represents the minimum probability of transitioning the agent from the current state to a safe state. For example, the robot can move from state (0, 1) to unsafe state (0,2) by taking action *right* so that the state safety value is 0 at state (0, 1). However, after taking action *down*, the robot can move from state (0, 1) to state (1, 1), which is a safe state. Consequently, the maximum action safety value is 1.00 at state (0, 1) as shown in Fig. 8(b).

We also consider the scenario with action uncertainties due to the actuator malfunction. Therefore, the considered MDP is stochastic. After an action is selected, the robot moves in the desired direction with a probability of 80%. On the other hand, it can go in each side direction with a probability of 10%. Figure 9 illustrates the estimated state values and state safety values after one simulation via the developed QSQ-learning. Theoretically, the state values, i.e., the maximal task-satisfaction probabilities, shall be 1.0, 0.9, or 0.8 at all non-unsafe states due to the action uncertainties. For example, the robot can take a *right* action at state (1, 2) to a safe absorbing

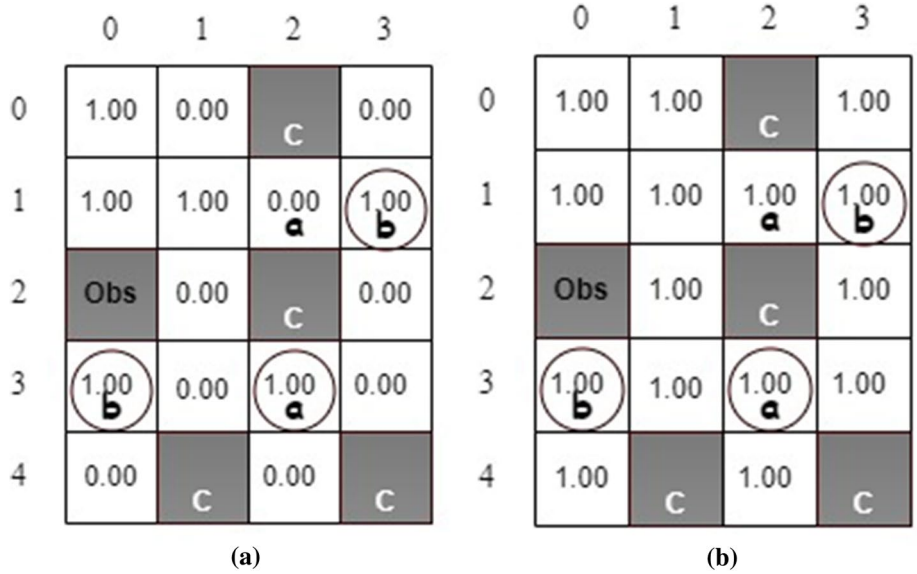

Figure 8. The results of (a) state safety values and (**b**) the maximum action safety values.

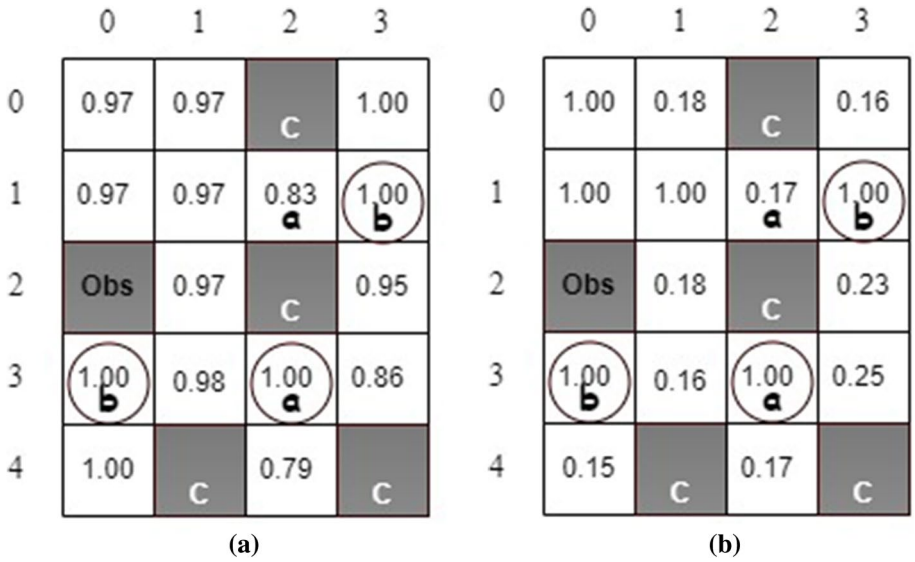

Figure 9. The estimated (**a**) state values and (**b**) state safety values for stochastic actions.

state, (1, 3). Therefore, the probability of the robot fulfilling the task requirement is 80%. The estimated state values in Fig. 9a from our simulation agree with the theoretical predictions.

On the other hand, the state safety value at state (1, 2) in Fig. 9b is expected to be 0.2 because the robot has a minimum probability of 20% of reaching a safe state after taking action *up* or *down* (the estimated safety value at this state is 0.17). One optimal policy is obtained, as shown in Fig. 10. The state values indicate at least one path generated from the optimal policy so that the robot can accomplish the task. We investigate the evolution of the times the robot visits unsafe states during the learning and compare the results with the one from Q-learning without safety value functions, as shown in Fig. 11. It can be seen that implementing safety value functions dramatically reduces the number of visits to unsafe states up to 87%.

Unlike a previous work[17], in which an agent was able to observe the safety statuses of its neighboring states, it is assumed that the agent can observe the current state's safety status only. Therefore, the agent must visit unsafe states due to exploration, especially at the beginning of learning, to obtain enough information for calculating the safety value functions. However, considering safety in action selection significantly saves the number of visits to unsafe states compared to the conventional RL methods.

Slippery grid-world. In this example, a robot moves on a 10×10 grid world, shown in Fig. 12, where the robot can "slip" to any adjacent state with a probability of 15% when taking action. The robot takes off from the

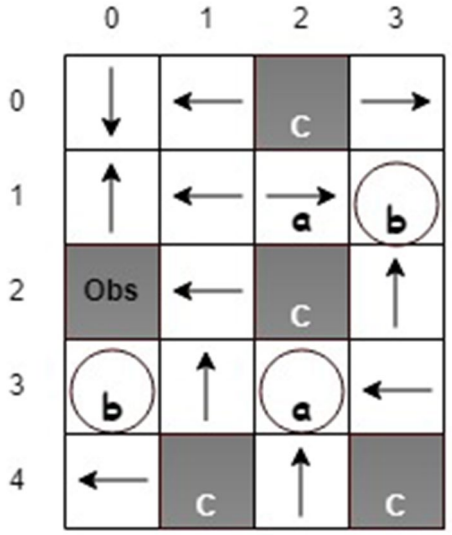

Figure 10. The generate optimal policy.

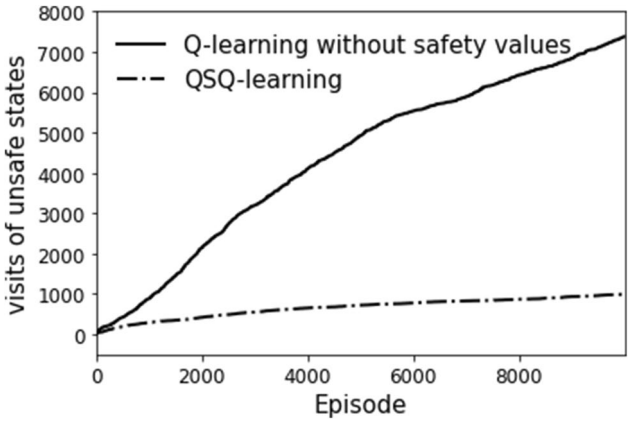

Figure 11. The evolution of visiting unsafe states.

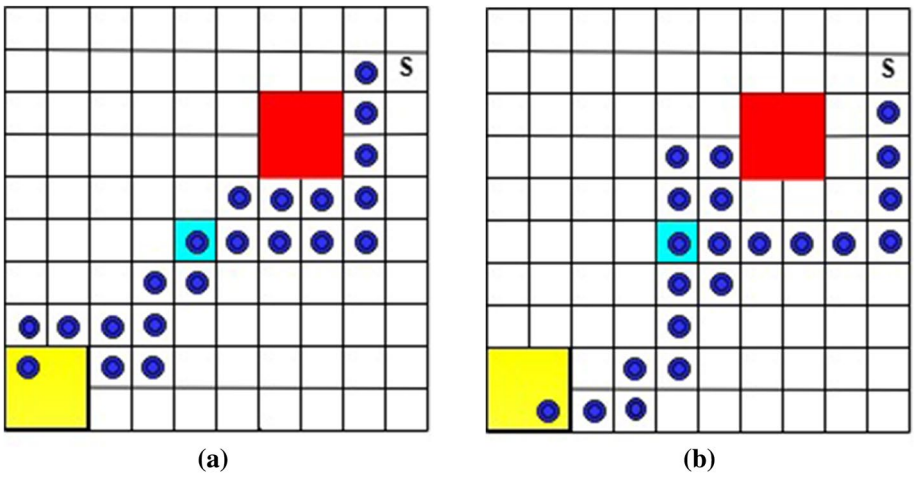

Figure 12. The generated trajectories from the optimal policy learned via Q-learning without safety value functions.

initial state (marked as "S" in Fig. 12) and tries to visit *goal*1 (cyan) and then *goal*2 (yellow) for infinite times while avoiding unsafe states (red). The task can be specified as the following LTL formula.

$$\varphi_2 = \square(goal1\ \mathcal{U}\ goal2) \wedge \square\neg\ unsafe \tag{32}$$

We conducted 10 simulations for both Q-learning (without safety value functions) and the developed QSQ-learning. Each simulation consists of 1000 episodes with 4000 steps per episode. Two trajectories, induced from the optimal policies via Q-learning, for the first round of visiting *goal*1 and then *goal*2 are illustrated in Fig. 12. In addition, Fig. 13 includes two trajectories generated from the optimal policies learned via QSQ-learning. For each learning method, we record the maximum and minimum times the robot visits unsafe states during the simulations and list them in Table 1. It can be seen that the robot visits unsafe states many fewer times when implementing unsafe value functions in learning.

Conclusions and future work

This paper presents safe reinforcement learning to find RL control policies that satisfy LTL specifications over finite and infinite horizons. The developed reward-shaping process further improves the reward density and guides the LTL satisfaction with maximum probability, while the safe padding technique utilizes the properties of E-LDGBA and maintains the safe exploration without influencing the original probabilistic guarantee. In addition, the quantum action selection technique provides an alternative approach to balancing exploration and exploitation during RL while taking advantage of quantum computing.

The state labels, including safety labels, are atomic propositions. In real-world problems, the agent can acquire the labels based on the collected information via perception sensors and its prior knowledge base. The safety value function proposed in this paper represents the probability of safely reaching the next state after taking a selected action. Such a concept can be extended to calculate the expected probability of reaching safe states after taking action and following the current policy within a finite or infinite horizon. In addition, the idea of

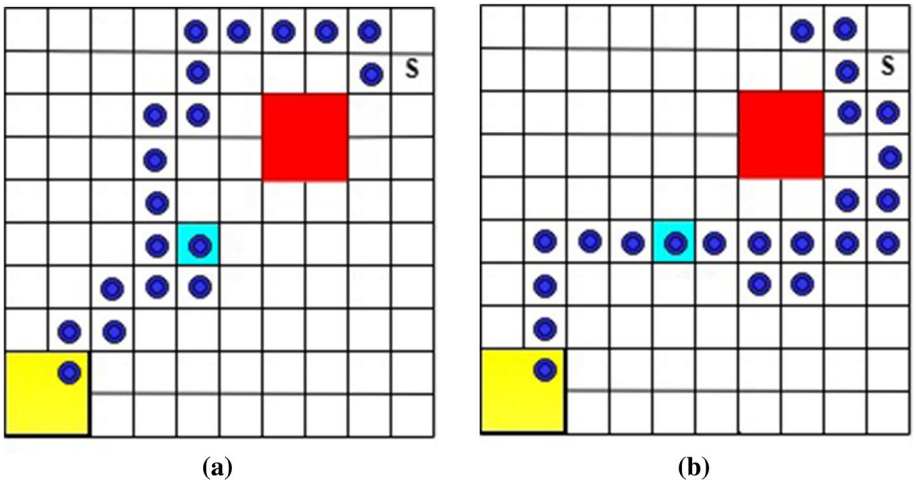

Figure 13. The generated trajectories from the optimal policy learned via QSQ-learning.

	maximum times	minimum times
Q-learning without safety values	160,937	157,192
QSQ-learning	2,294	962

Table 1. The number of times the robot visits unsafe states during learning.

calculating safety value functions for a continuous state space, proposed as Equations (22) and (23), will be refined in future studies.

Quantum computing is powerful because the quantum algorithms, such as Grover's search algorithm, have lower computational complexities than their classical equivalents. However, current quantum computers are relatively small (up to 433 qubits in the largest quantum computer built by IBM) and noisy (not fault-tolerant). Indeed, we are in the era of Noisy Intermediate-Scale Quantum (NISQ)[45]. Consequently, Variational Quantum-Classical (VQC) algorithms have become popular in deploying quantum algorithms on near-term quantum devices. In VQC algorithms, classical computers perform the overall computation task on information they acquire from running calculations on a quantum computer. Our quantum safe RL approach adopts the same strategy: the learning process is conducted on a classical computer while the action is selected via quantum computing. The proposed quantum action selection algorithm results in parameterized quantum circuits, which are relatively small, short-lived, and thus suitable for NISQ computers. Although quantum action selection is performed on the quantum simulator in our simulations, it can be run on a quantum machine as the backend (see Fig. 4).

Although quantum computing theoretically promises to be exponentially faster than classical computers, we don't think the computation would be faster for our simulation examples, even using a quantum machine to conduct action selection because the action spaces are small. However, we expect our method to take advantage of quantum physics's fundamental properties, especially superposition when handling problems with large/continuous action spaces. For example, our previous work[1] studied motion planning for Mars exploration cases with a large-scale continuous environment. Our method can be utilized to solve the same problems by applying a fine-scale discretization to the action space since an n-qubit system can represent 2^n actions. The method can be extended to the discretization of a continuous state space with quantum states.

In addition, the proposed quantum action selection method can also be applied to policy-based RL methods, in which the actions are predicted with various probabilities. In this case, the basic quantum state corresponding to the action with the highest predicted probability would be marked. Considering the above-mentioned motivations and challenges, further research is needed to extend this method to problems with continuous state and action spaces.

Data availability

The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

Received: 30 June 2022; Accepted: 20 January 2023
Published online: 02 February 2023

References

1. Cai, M., Hasanbeig, M., Xiao, S., Abate, A. & Kan, Z. Modular deep reinforcement learning for continuous motion planning with temporal logic. *IEEE Robot. Autom. Lett.* **6**(4), 7973–7980 (2021) arXiv:2102.12855.
2. Baier, C. & Katoen, J.-P. *Principles of model checking* (The MIT Press, Cambridge, 2008).
3. Guo, M. & Zavlanos, M. M. Probabilistic motion planning under temporal tasks and soft constraints. *IEEE Trans. Autom. Control* **63**(12), 4051–4066 (2018).
4. Cai, M., Li, Z., Gao, H., Xiao, S., & Kan, Z. Optimal Probabilistic Motion Planning with Potential Infeasible LTL Constraints. *IEEE Trans. Automat. Control* **68**(1), 301–316 (2023).
5. Sutton, R. S. & Barto, A. G. *Reinforcement Learning: An Introduction* (The MIT Press, Cambridge, 2018).
6. Garcıa, J. & Fernández, F. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.* **16**(1), 1437–1480 (2015).
7. Moldovan, T.M., & Abbeel, P. Safe exploration in markov decision processes. arXiv preprint arXiv:1205.4810 (2012).
8. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., & Topcu, U. Safe reinforcement learning via shielding. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018).
9. Cheng, R., Orosz, G., Murray, R.M., & Burdick, J.W. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 3387–3395 (2019).
10. Wen, M. & Topcu, U. Constrained cross-entropy method for safe reinforcement learning. *IEEE Trans. Autom. Control* **66**(7), 3123–3127. https://doi.org/10.1109/TAC.2020.3015931 (2021).
11. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., & Wojtczak, D. Omega-regular objectives in model-free reinforcement learning. In: Int. Conf. Tools Alg. Constr. Anal. Syst., pp. 395–412 (2019). Springer
12. Cai, M., Peng, H., Li, Z. & Kan, Z. Learning-based probabilistic LTL motion planning with environment and motion uncertainties. *IEEE Trans. Autom. Control* **66**(5), 2386–2392. https://doi.org/10.1109/TAC.2020.3006967 (2021).
13. Hasanbeig, M., Kantaros, Y., Abate, A., Kroening, D., Pappas, G.J., Lee, I.: Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In: Proc. IEEE Conf. Decis. Control, pp. 5338–5343 (2019). IEEE
14. Bozkurt, A.K., Wang, Y., Zavlanos, M.M., & Pajic, M. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In: Int. Conf. Robot. Autom., pp. 10349–10355 (2020). IEEE.
15. Cai, M., Xiao, S., Li, B., Li, Z., & Kan, Z. Reinforcement learning based temporal logic control with maximum probabilistic satisfaction. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 806–812 (2021). https://doi.org/10.1109/ICRA48506.2021.9561903.
16. Li, X., Serlin, Z., Yang, G. & Belta, C. A formal methods approach to interpretable reinforcement learning for robotic planning. *Sci. Robot.***4**(37), (2019).
17. Hasanbeig, M., Abate, A., & Kroening, D. Cautious reinforcement learning with logical constraints. AAMAS'20: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, 483–491 (2020).
18. Sickert, S., Esparza, J., Jaax, S., & Křetínskỳ, J. Limit-deterministic Büchi automata for linear temporal logic. In: Int. Conf. Comput. Aided Verif., pp. 312–332 (2016). Springer.
19. Nielsen, M.A., & Chuang, I.L. Quantum Computation and Quantum Information, 10th edn. Cambridge University Press, New York (2010). https://doi.org/10.1017/CBO9780511976667.
20. Biamonte, J. *et al.* Quantum machine learning. *Nature* **549**(7671), 195–202. https://doi.org/10.1038/nature23474 (2017).
21. Beer, K. *et al.* Training deep quantum neural networks. *Nat. Commun.* **11**(1), 1–6. https://doi.org/10.1038/s41467-020-14454-2 (2020).
22. Cong, I., Choi, S. & Lukin, M. D. Quantum convolutional neural networks. *Nat. Phys.* **15**(12), 1273–1278. https://doi.org/10.1038/s41567-019-0648-8 (2019).
23. Iyengar, S.S., Kumar, L.K.J., & Mastriani, M. Analysis of five techniques for the internal representation of a digital image inside a quantum processor (2020) arXiv:2008.01081.
24. Li, Y., Zhou, R.-G., Xu, R., Luo, J. & Hu, W. A quantum deep convolutional neural network for image recognition. *Quantum Sci. Technol.* **5**(4), 044003. https://doi.org/10.1088/2058-9565/AB9F93 (2020).
25. Hu, W., Hu, J., Hu, W. & Hu, J. Reinforcement learning with deep quantum neural networks. *J. Quantum Inf. Sci.* **9**(1), 1–14. https://doi.org/10.4236/JQIS.2019.91001 (2019).
26. Denchev, V. S. *et al.* What is the computational value of finite-range tunneling?. *Phys. Rev. X* **6**(3), 031015. https://doi.org/10.1103/PhysRevX.6.031015 (2016).
27. Saggio, V. *et al.* Experimental quantum speed-up in reinforcement learning agents. *Nature* **591**(7849), 229–233. https://doi.org/10.1038/s41586-021-03242-7 (2021).
28. Dong, D., Chen, C., Li, H. & Tarn, T. J. Quantum reinforcement learning. *IEEE Trans. Syst. Man Cybern. B Cybern.* **38**(5), 1207–1220. https://doi.org/10.1109/TSMCB.2008.925743 (2008).
29. Ganger, M. & Hu, W. Quantum multiple Q-learning. *Int. J. Intell. Sci.* **09**(01), 1–22. https://doi.org/10.4236/IJIS.2019.91001 (2019).
30. Fernandez-Gauna, B., Graña, M., Lopez-Guede, J. M., Etxeberria-Agiriano, I. & Ansoategui, I. Reinforcement Learning endowed with safe veto policies to learn the control of Linked-Multicomponent Robotic Systems. *Inf. Sci.* **317**, 25–47. https://doi.org/10.1016/J.INS.2015.04.005 (2015).
31. Fulton, N. & Platzer, A. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. *Proc. AAAI Conf. Artif. Intell.* **32**(1), 6485–6492. https://doi.org/10.1609/AAAI.V32I1.12107 (2018).
32. Wootters, W. & Zurek, W. A single quantum cannot be cloned. *Nature* **299**, 802–803. https://doi.org/10.1038/299802a0 (1982).
33. Watkins, C. J. & Dayan, P. Q-learning. *Mach. Learn.* **8**(3–4), 279–292 (1992).
34. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M.A. Playing atari with deep reinforcement learning. CoRR (2013) arXiv:1312.5602.
35. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. Proximal policy optimization algorithms. arXiv:1707.06347 (2017) [cs.LG].
36. Kretínský, J., Meggendorfer, T., & Sickert, S. Owl: A library for ω-words, automata, and LTL. In: Autom. Tech. Verif. Anal., pp. 543–550 (2018). https://doi.org/10.1007/978-3-030-01090-4_34.
37. Sadigh, D., Kim, E.S., Coogan, S., Sastry, S.S., & Seshia, S.A. A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In: Proc. IEEE Conf. Decis. Control., pp. 1091–1096 (2014).
38. Ng, A.Y., Harada, D., & Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In: ICML, vol. 99, pp. 278–287 (1999).
39. Dempster, A. P., Laird, N. M. & B, R. D. Maximum likelihood from incomplete data via the em algorithm. *J. Roy. Stat. Soc.: Ser. B (Methodol.)* **39**(1), 1–22 (1977).
40. Bertsekas, D. P. & Tsitsiklis, J. N. *Neuro-dynamic Programming* Vol. 1 (Athena scientific, Belmont, MA, 1996).
41. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 1928–1937. PMLR, New York, New York, USA (2016). https://proceedings.mlr.press/v48/mniha16.html.
42. Grover, L. K. Quantum mechanics helps in searching for a needle in a Haystack. *Phys. Rev. Lett.* **79**(2), 325. https://doi.org/10.1103/PhysRevLett.79.325 (1997).

43. Koch, D., Wessing, L., & Alsing, P.M. Introduction to Coding Quantum Algorithms: A Tutorial Series Using Qiskit (2019) arXiv: 1903.04359.
44. Qiskit. https://qiskit.org/ Accessed 2021-08-10.
45. Preskill, J. Quantum Computing in the NISQ era and beyond. *Quantum* **2**, 79. https://doi.org/10.22331/q-2018-08-06-79 (2018).

Author contributions

All authors contributed to the study conception and design. Problem definitions, theory approvals, and simulations were performed by M.C and S.X. The first draft of the manuscript was written by M.C and S.X. All authors commented on previous versions of the manuscript. All authors read and approved the final manuscript. All authors have approved the manuscript and agreed with its publication on the Journal of Intelligent & Robotic Systems.

Funding Information

Xiao would like to thank the US Department of Education (ED#P116S210005) for supporting this research.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to S.X.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.