Tractable Reinforcement Learning for Signal Temporal Logic Tasks With Counterfactual Experience Replay

Siqi Wang, *Graduate Student Member, IEEE*, Xunyuan Yin, Shaoyuan Li, *Senior Member, IEEE*, and Xiang Yin, *Member, IEEE*

***Abstract*—We investigate the control synthesis problem for Markov decision processes (MDPs) with unknown transition probabilities under signal temporal logic (STL) specifications. Our primary objective is to learn a control policy that maximizes the probability of satisfying the STL task. However, existing approaches to STL control synthesis using reinforcement learning encounter a significant exploration challenge, particularly when expanding the state space to incorporate STL tasks. In this letter, we propose a novel reinforcement learning algorithm tailored for STL tasks, addressing the exploration difficulty by effectively leveraging *counterfactual experiences* to expedite the training process. Through experiments we show that these generated experiences enable us to fully employ the knowledge embedded within the task, resulting in a substantial reduction in the number of trial-and-error explorations required before achieving convergence.**

***Index Terms*—Reinforcement learning, signal temporal logic, formal methods.**

I. Introduction

With the rapid advancement of cyber-physical systems (CPS), there is a growing imperative to formally verify and synthesize spatial-temporal behaviors and provide provable guarantees. As a formal specification language, signal temporal logic (STL) offers a structured approach to reasoning about temporal properties related to real-valued physical signals. STL empowers the expression of intricate requirements, such as "drive to the charging station within 20 minutes and remain there for at least 10 minutes." In the past years, STL has been successfully applied to various safety-critical CPSs such as autonomous vehicles [1], industrial automation [2], smart grids [3] and healthcare systems [4].

To address the control synthesis problem concerning STL specifications, recent literature has witnessed the development of various synthesis methods. Effective strategies include encoding the satisfaction of STL formulae as constraints using mixed-integer linear programming techniques [5], [6], [7] and encapsulating the forward invariant satisfaction regions of STL tasks using control barrier functions [8], [9], [10]. However, these approaches necessitate complete knowledge of the system's dynamics, which is not readily accessible in practical applications.

When the dynamics of the system are unknown a priori, model-free reinforcement learning (RL) becomes a prevalent approach for synthesizing control policies [11], [12], [13], [14], [15], [16], [17]. In the context of RL for STL, a prominent approach is the τ-MDP approach [11]. However, this approach heavily relies on partial history trajectories, leading to an exponential growth in state space with the maximum sub-task horizon. In [15], the authors proposed an F-MDP formulation to more efficiently record historical information. However, even with this formulation, the state space can still become excessively large in complex scenarios. Therefore, substantial interactions between the agent and the environment are essential to generate sufficient experiences for effective learning, which may be time-consuming and mechanically harmful.

Experience replay is a widely utilized technique in RL that can enhance sampling efficiency by storing and reusing past experiences. Through random sampling from these experiences during training, the agent can disrupt temporal correlations and improve generalization [18]. Our research draws inspiration from [19], [20], wherein counterfactual experience replay is incorporated into policy synthesis for linear temporal logic (LTL) specifications. This approach leverages the known automaton representation of the LTL task, incorporating that information to generate counterfactual experiences. However, to our knowledge, leveraging counterfactual experiences to expedite RL for STL tasks has not been explored. This problem presents a significant departure from the LTL problem, as STL lacks similar automata representations.

Motivated by the necessity for a more efficient RL algorithm tailored for STL tasks, this letter adopts the philosophy of counterfactual experience replay to guide RL for STL synthesis. Specifically, our contributions can be summarized as follows. First, we utilize the F-MDP model to encode both the physical state and the task progress state of the

Manuscript received 5 March 2024; accepted 21 April 2024. Date of publication 7 May 2024; date of current version 28 May 2024. This work was supported by the National Natural Science Foundation of China under Grant 62173226 and Grant 62061136004. Recommended by Senior Editor A. P. Aguiar. *(Corresponding author: Xiang Yin.)*

Siqi Wang, Shaoyuan Li, and Xiang Yin are with the Department of Automation and the Key Laboratory of System Control and Information Processing, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: yinxiang@sjtu.edu.cn).

Xunyuan Yin is with the School of Chemistry, Chemical Engineering and Biotechnology, Nanyang Technological University, Singapore.

Digital Object Identifier 10.1109/LCSYS.2024.3397844

agent. Building upon this F-MDP representation, we introduce an approach to generate counterfactual experiences from factual ones. Subsequently, we present an off-policy RL framework designed to synthesize policies for STL-specified tasks, effectively leveraging the generated information. Finally, we conduct an extensive set of experiments, focusing on a path planning problem for a mobile robot within a factory floor. We thoroughly analyze the efficiency of our approach in comparison to the baseline methodologies.

II. Preliminaries

A. Signal Temporal Logic Tasks

We consider tasks specified by *signal temporal logic* (STL) formulae. STL is a formal language widely used for describing high-level logic behaviors over continuous signals. Similar to the setting of [15], we consider a restrictive yet expressive fragment of STL with the following syntax:

$$\begin{aligned}\Phi &:= \neg\Phi \mid \mathbf{F}_{[0,T)}\phi,\\ \phi &:= \neg\phi \mid \phi_1\wedge\phi_2 \mid \mathbf{F}_{[0,\tau)}\psi,\\ \psi &:= \mathsf{true} \mid \mu \mid \neg\psi \mid \psi_1\wedge\psi_2,\end{aligned}\tag{1}$$

where $\mu : \mathbb{R}^m \to \{\mathsf{true}, \mathsf{false}\}$ is an atomic predicate such that it is satisfied for signal value $s \in \mathbb{R}^m$ iff $h^\mu(s) > 0$, where $h^\mu : \mathbb{R}^m \to \mathbb{R}$ is the underlying predicate function; $\neg$ and $\wedge$ are the Boolean operators "negation" and "conjunction", respectively; $\mathbf{F}$ is the temporal operator "eventually" with $\tau, T \in \mathbb{N}_{\geq 0}$ are two non-negative time instants.

Intuitively, formula class ψ represents Boolean formulae, in which "disjunction" $\vee$ and "implication" $\rightarrow$ can also be induced. Formula class ϕ represents the sub-formulae, in which temporal operators can only be applied in front of a Boolean formula ψ. Note that, within this class, one can also define the temporal operator "always" $\mathbf{G}_{[0,\tau)}\phi := \neg\mathbf{F}_{[0,\tau)}\neg\phi$. Finally, formula class Φ captures the overall task of our interest, in which two nested temporal operators are allowed in front of Boolean formulae.

STL formulae are evaluated over finite signals that take values in a continuous metric space $\mathbb{R}^m$. For any signal $\mathbf{s}$, we denote by s_t its value at time t. Also, we denote by $(\mathbf{s}, t) \models \varphi$ that signal $\mathbf{s}$ satisfies formula φ at time instant t. Formally, the (Boolean) semantic of STL is defined recursively as

$$\begin{aligned}(\mathbf{s},t)\models\mu &\Leftrightarrow h^\mu(s_t)>0,\\ (\mathbf{s},t)\not\models\varphi &\Leftrightarrow (\mathbf{s},t)\models\neg\varphi,\\ (\mathbf{s},t)\models\varphi_1\wedge\varphi_2 &\Leftrightarrow (\mathbf{s},t)\models\varphi_1 \text{ and } (\mathbf{s},t)\models\varphi_2,\\ (\mathbf{s},t)\models\mathbf{F}_{[0,T)}\varphi &\Leftrightarrow \left(\mathbf{s},t'\right)\models\varphi\ \exists t'\in[t,t+T).\end{aligned}\tag{2}$$

Based on the above semantics, the semantic for the temporal operator "always" $\mathbf{G}_{[0,T)}$ can also be induced as

$$(\mathbf{s},t)\models\mathbf{G}_{[0,T)}\varphi \Leftrightarrow \left(\mathbf{s},t'\right)\models\varphi\ \forall t'\in[t,t+T).$$

We also define the *characteristic function* of an STL formula φ w.r.t. signal $\mathbf{s}$ at time instant t by

$$\chi(\varphi,\mathbf{s},t)=\begin{cases}1 \text{ if } & (\mathbf{s},t)\models\varphi,\\ 0 \text{ otherwise,}\end{cases}\tag{3}$$

where $\chi(\varphi, \mathbf{s}, t) = \chi(\varphi, \mathbf{s})$ when $t = 0$. For any STL formula, its satisfaction can be completely determined within its *horizon*, denoted by $\mathrm{hrz}(\varphi)$, which can be computed as the maximum sum of the time interval bound of all nested temporal operators [11].

B. Q-Learning for Markov Decision Processes

We model the dynamic of the underlying system by a Markov decision process (MDP). Formally, an MDP is defined as a 5-tuple $M = (\Sigma, s_0, A, P, R)$, where Σ is the set of states, s_0 is the initial state, A is the set of actions, $P : \Sigma \times A \times \Sigma \to [0, 1]$ is a *unknown* transition probability function and $R : \Sigma \to \mathbb{R}$ is the reward function. Note that we include the reward function R to maintain alignment with standard models despite it is not directly used in our current method. For simplicity, we focus on a single initial state; all results can be extended easily to the case of initial distribution.

Given MDP M, a (stationary) *control policy* is a function $\pi : \Sigma \times A \to [0, 1]$ that assigns each action a probability at each state with $\forall s \in \Sigma : \sum_{a\in A}\pi(s, a) = 1$. The objective is to synthesize a control policy that maximizes the total discounted rewards [18], i.e.,

$$\pi^*=\arg\max_\pi \mathbb{E}\left[\sum_{t=0}^{T-1}\gamma^t r_t\right],\tag{4}$$

where r_t is the random variable for the reward at instant t when the policy π is applied and $\gamma \in [0, 1]$ is a discount factor that penalizes the reward in the future.

Since the transition probability is assumed to be unknown, one needs to *learn* the optimal control policy based on the online information of states, actions and rewards. Particularly, tabular Q-learning, one of the simplest yet widely used RL algorithms, is a model-free, temporal-difference method which maintains the values of all state-action pairs in a Q-table. Specifically, at each update, the Q-table is updated with a *transition* (s_t, a_t, r_t, s_{t+1}) according to the Bellman equation:

$$Q(s_t,a_t):=(1-\alpha)Q(s_t,a_t)+\alpha\left[r_t+\gamma\max_{a\in A}Q(s_{t+1},a)\right],\tag{5}$$

where α is the learning rate.

III. Problem Formulation

Given an MDP $M = (\Sigma, s_0, A, P, R)$ with unknown P and STL formula Φ, the overall control objective considered in this letter is to maximize the probability of the satisfaction of STL formula Φ. Formally, we aim to find the following optimal policy π^* defined by

$$\pi^*=\arg\max_\pi \mathbb{P}^\pi[(\mathbf{s},0)\models\Phi],\tag{6}$$

where $\mathbb{P}^\pi$ is the probability measure over the set of all finite signals of length $\mathrm{hrz}(\Phi)$ under policy π. Note that, using the characteristic function, Equation (6) can also be written equivalently as

$$\pi^*=\arg\max_\pi \mathbb{E}^\pi[\chi(\Phi,\mathbf{s})].\tag{7}$$

Note that the objective function in the above formulation cannot be handled by the standard Q-learning algorithm since the reward is defined over the entire horizon rather than in the discounted sum form. Therefore, following the approach in [11], we use *Log-Sum-Exp* (LSE) to transform the original objective into the RL-friendly form. Specifically, the LSE is a smooth approximation method for max and min operators:

$$\max(x_1,\ldots,x_n)\approx\frac{1}{\beta}\log\sum_{i=1}^n e^{\beta x_i},$$

$$\min(x_1, \ldots, x_n) \approx -\frac{1}{\beta}\log\sum_{i=1}^{n} e^{-\beta x_i}, \tag{8}$$

where the approximation error is bounded by two inequalities, see [11].

In the context of the STL control synthesis problem, since we focus on the STL formula of form $\mathbf{F}_{[0,T)}\phi$ or $\mathbf{G}_{[0,T)}\phi$, the control synthesis problem stated in Equation (6) can be approximated as follows:

$$\pi_A^* = \arg\max_{\pi} \mathbb{E}\begin{cases} \sum_{t=0}^{T-1} e^{\beta\chi(\phi,\mathbf{s},t)} & \text{if } \Phi = \mathbf{F}_{[0,T)}\phi, \\ \sum_{t=0}^{T-1} -e^{-\beta\chi(\phi,\mathbf{s},t)} & \text{if } \Phi = \mathbf{G}_{[0,T)}\phi. \end{cases} \tag{9}$$

In the above problem reformulation, the objective is in form of a summation of rewards, where each step reward can be determined within the horizon of sub-formula ϕ. In order to calculate the step reward, a direct approach is to augment the original MDP to keep track of all states in the latest hrz(ϕ)-step history. This is known as the τ-MDP approach [11], which results in an augmented state space of size equal to $|\Sigma|^{\text{hrz}(\phi)}$. However, this approach requires extremely large memory when the horizon of ϕ increases. In this letter, we will introduce a simple and tractable method for calculating sub-task satisfaction.

IV. Reinforcement Learning With Counterfactual Experience Replay

In this section, we introduce our main algorithm for approximating the objective function, building upon the flag-based encoding of task progress proposed in [15]. Additionally, we utilize the STL formula to guide the generation of counterfactual experiences, enhancing the learning process and resulting in a more efficient approach compared to the original algorithm in [15].

A. Tractable Representation of Task Progress

First, we recall the F-MDP defined in [15] that captures the system state as well as the task progress. The definition is slightly different from the original version for the purpose of later developments. Recall that our task is of form $\mathbf{F}_{[0,T)}\phi$ or $\mathbf{G}_{[0,T)}\phi$. Furthermore, each formula ϕ is a Boolean combination of sub-formulae. Therefore, we can write ϕ in terms of its sub-formulae by $\phi = g(\phi_1, \ldots, \phi_n)$, where g is a Boolean function and each ϕ_i is of form $\mathbf{F}_{[0,\tau_i)}\psi_i$ or $\mathbf{G}_{[0,\tau_i)}\psi_i$. Therefore, an F-MDP is constructed by augmenting the state space of the original MDP by assigning flag variables for each sub-formula ϕ_i to keep track of its progress.

Definition 1 (F-MDP): For MDP $M = (\Sigma, s_0, A, P, R)$, its F-MDP is defined as a tuple $M^F = (\Sigma^F, s_0^F, A, P^F, R^F)$, where

- $\Sigma^F \subseteq (\Sigma \times \prod_{i=1}^{n} \mathfrak{F}_i)$ is the augmented state space obtained by taking the Cartesian product of the system state space with the n flag state sets $\mathfrak{F}_i, i \in \{1, \ldots, n\}$;
- $s_0^F = (s_0, f_{1,0}, \ldots, f_{n,0})$ is the initial state with $f_{i,0} = 0, \forall i \in \{1, \ldots, n\}$;
- $P^F : \Sigma^F \times A \times \Sigma^F \to [0, 1]$ is the transition function. Let $s^F = (s, f_1, f_2, \ldots, f_n)$ and $s^{F'} = (s', f_1', f_2', \ldots, f_n')$. We have $P^F(s^F, a, s^{F'}) = P(s, a, s')$ if and only if $f_i' = update(f_i, s'), \forall i \in \{1, \ldots, n\}$, where $update(\cdot)$ is the update rule of flag variables defined by:

$$f_{i,t+1} = \begin{cases} \tau_i & \text{if } s_{t+1} \models \psi_i \;\&\; \phi_i = \mathbf{F}_{[0,\tau_i)}\psi_i, \\ \max(f_{i,t} - 1, 0) & \text{if } s_{t+1} \not\models \psi_i \;\&\; \phi_i = \mathbf{F}_{[0,\tau_i)}\psi_i, \\ \min(f_{i,t} + 1, \tau_i) & \text{if } s_{t+1} \models \psi_i \;\&\; \phi_i = \mathbf{G}_{[0,\tau_i)}\psi_i, \\ 0 & \text{if } s_{t+1} \not\models \psi_i \;\&\; \phi_i = \mathbf{G}_{[0,\tau_i)}\psi_i. \end{cases} \tag{10}$$

- $R^F : \Sigma^F \to \mathbb{R}$ is a reward function defined over the augmented state space.

The update rule of the flag variables is the essence of the F-MDP formulation. Each flag variable captures the completion progress of each sub-task and one can use it to determine the overall satisfaction. For each sub-task, the satisfaction of each sub-task is dependent on how $s_t \models \psi_i$ is seen in the last τ_i-step history. For $\phi_i = \mathbf{F}_{[0,\tau_i)}\psi_i, f_{i,t} > 0$ means that there exists at least one t' in the last τ_i-step history that $s_{t'} \models \psi_i$ holds. And for $\phi_i = \mathbf{G}_{[0,\tau_i)}\psi_i, f_{i,t} = \tau_i$ means there exists continuous τ_i-step satisfaction of $s_{t'} \models \psi_i$. Thereby we derive satisfaction for sub-task ϕ_i:

$$\chi(\phi_i, \mathbf{s}_t^F) = \begin{cases} 1 & \text{if } (f_{i,t} > 0 \;\&\; \phi_i = \mathbf{F}_{[0,\tau_i)}\psi_i) \\ & \text{or } (f_{i,t} = \tau_i \;\&\; \phi_i = \mathbf{G}_{[0,\tau_i)}\psi_i) \\ 0 & \text{else} \end{cases} \tag{11}$$

The satisfaction of the inner formula ϕ can be derived using simple Boolean operations according to Equation (2). Thus the step reward for the agent can be derived from Equation (11). Specifically, the step reward is given by:

$$r_t = R^F\left(s_{t+1}^F\right) = \begin{cases} e^{\beta\chi\left(\phi,\mathbf{s}_{t+1}^F\right)} & \text{if } \Phi = \mathbf{F}_{[0,T)}\phi \\ -e^{-\beta\chi\left(\phi,\mathbf{s}_{t+1}^F\right)} & \text{if } \Phi = \mathbf{G}_{[0,T)}\phi \end{cases}. \tag{12}$$

One can easily derive that the size of the state space for F-MDP is $|\Sigma| \cdot \prod_{i=1}^{n} \tau_i$.

Remark 1: In most circumstances, the number of sub-task, n, is often trivial compared to each sub-task's horizon τ_i, $\forall i \in \{1, \ldots, n\}$, which can increase drastically as the problem complicates when one sub-task takes longer to finish. The F-MDP representation is applicable to a broader context. Only in extreme circumstances, where the inner formula consists of multiple sub-formulas with short horizons, and the original state space is small, the τ-MDP representation is more economic.

B. STL-Guided Counterfactual Experience Generation

Although the F-MDP formulation reduces the state space to certain degree, the state space size still grows exponentially with the number of sub-tasks n and linearity with each sub-task horizon hrz(ϕ_i). In RL processes, experiences are obtained by interacting with physical systems or simulation environments. Optimal policies can be learned given all states are visited infinitely many times. However, when there exist multiple sub-tasks or long-horizon sub-tasks, it is extremely time-and-resource-consuming to explore the entire state space.

In order to enhance the usage of samples, we propose an STL-guided counterfactual experience generation scheme to combat the issue of large state space. The philosophy of our experience generation procedure is simple. Specifically, in the F-MDP constructed, at each time step t, the transition of

Algorithm 1 Counterfactual Experience Replay (CFER)

```
1: Initialize the learning algorithm $\mathbb{A}$    ▷ Q-Learning
2: Initialize the replay buffer $\mathcal{B}$
3: for episode $= 1 : M$ do
4:     Sample an initial state $s_0$.
5:     for $t = 0, H-1$ do
6:         Interact and get $e = (s_t, \mathbf{f}_t, a_t, r_t, s_{t+1}, \mathbf{f}_{t+1})$
7:         Generate counterfactual transitions $\tilde{E}$ via (14)
8:         Store the transitions in $\mathcal{B}$
9:         if episode%Replay Period$= 0$ then
10:            Sample a minibatch $B := \mathbb{S}(\mathcal{B})$
11:            Update policy $\pi := \mathtt{UPDATE}_{\mathbb{A}}(\pi, B)$
12:        end if
13:    end for
14: end for
15: return Policy $\pi$
```

flag variable is fully known given the resulted signal state s_{t+1}. Therefore, in addition to considering the original flag transition, one can further use this physical transition sampled together with artificial flags to generate new associated flag transitions.

To be more specific, at each step t, the agent takes an action a_t from state s_t^F and then lands in state s_{t+1}^F, receiving a reward r_t. For the convenience of further discussions, we write $s_t^F = (s_t, f_{1,t}, \dots, f_{n,t})$ and combine the flag states $(f_{1,t}, \dots, f_{n,t})$ to form a vector $\mathbf{f}_t \in \prod_{i=1}^n \mathfrak{F}_i$, where $\mathfrak{F}_i$ is the state set for each flag variable $f_{i,t}$. We define such transitions resulted from actual interactions with the physical system *factual* experiences. We store this experience as a tuple

$$e = (s_t, \mathbf{f}_t, a_t, r_t, s_{t+1}, \mathbf{f}_{t+1}). \quad (13)$$

Definition 2 (Counterfactual Experience): The set of counterfactual experiences generated by a factual experience e is defined by

$$\tilde{E} = \left\{ \left(s_t, \tilde{\mathbf{f}}_t, a_t, \tilde{r}_t, s_{t+1}, \tilde{\mathbf{f}}_{t+1} \right) \mid \forall \tilde{\mathbf{f}}_t \in \mathfrak{F} \right\}, \quad (14)$$

where $\tilde{\mathbf{f}}_{t+1} = update(s_{t+1}, \tilde{\mathbf{f}}_t)$ and $\tilde{r}_t = R^F(\tilde{s}_{t+1}^F), \tilde{s}_{t+1}^F = (s_{t+1}, \tilde{\mathbf{f}}_{t+1})$ as defined Equation (10) and (12).

Intuitively, from each factual experience, we can pretend that the agent starts from any arbitrary flag state $\mathbf{f}_t$ and generates a set of counterfactual experiences. In essence, to achieve complete coverage of the augmented state space, the agent is only required to explore the primary state space Σ, while the augmented states can be systematically generated.

In Algorithm 1, we provide a framework for RL with STL-guided counterfactual experience replay. During each learning step, the agent produces a factual experience and generates a set of counterfactual experiences according to Equation (14), which are then stored into the replay memory. At each replay period, the agent samples a batch of experiences from the memory based on a sampling strategy $\mathbb{S}$ and performs batch policy update according to Equation (5).

V. Case Study and Numerical Comparisons

In this section, we conduct a case study to illustrate the feasibility and effectiveness of the proposed algorithm. Specifically, we consider the control synthesis problem for a mobile robot navigating in a workspace. We show in the simulations that control policies can be learned to successfully achieve the STL tasks. Furthermore, we conduct a set of numerical experiments with different tasks and different grid sizes to compare the proposed algorithm with existing algorithms.

All algorithms are implemented using `Python 3.7` on a Ubuntu 20.04 machine. Simulations are provided by robot simulator `Coppeliasim`. Codes and video of sample trajectories are available at https://github.com/WSQsGithub/STL-CFER.

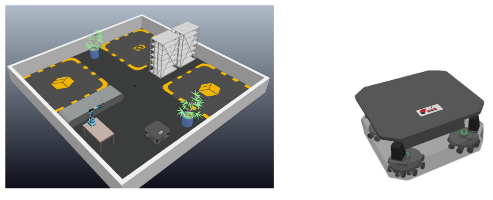
(a) Factory workspace. (b) Agent.

Fig. 1. Simulation environment.

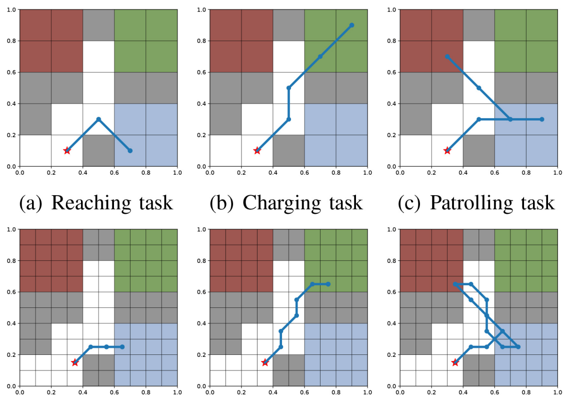
(a) Reaching task (b) Charging task (c) Patrolling task
(d) Reaching task (e) Charging task (f) Patrolling task

Fig. 2. Sample traces generated by optimal `CFER` policies.

A. Environment Description and STL Tasks

We consider the scenario, where a mobile robot navigates in a manufacturing factory as shown in Figure 1. As shown in the figure, the factory workspace has two regions where the robot can pick up or unload products and one charging region where the robot can charge its battery. The robot is navigating in the workspace in order to achieve some STL task. We discretize the workspace into a grid world, considering resolutions of 5×5 and 10×10 as shown in Figure 2. Then at each instant, the agent makes a high-level decision in the discrete abstract grid world to move to any of its adjacent grid or to stay still. If the agent chooses an action that results in collision with the boundaries, then it is forced to remain still in the same grid. Then the high-level command from grid to grid is executed by a low-level hybrid controller for navigation. Due to discretization errors as well as disturbances in the real world, the transitions from grid to grid under high-level actions are uncertain with an unknown probability.

We consider the following three different types of STL tasks, where signal s at each instant is the coordinate (x, y) of the center of the current grid of the robot:

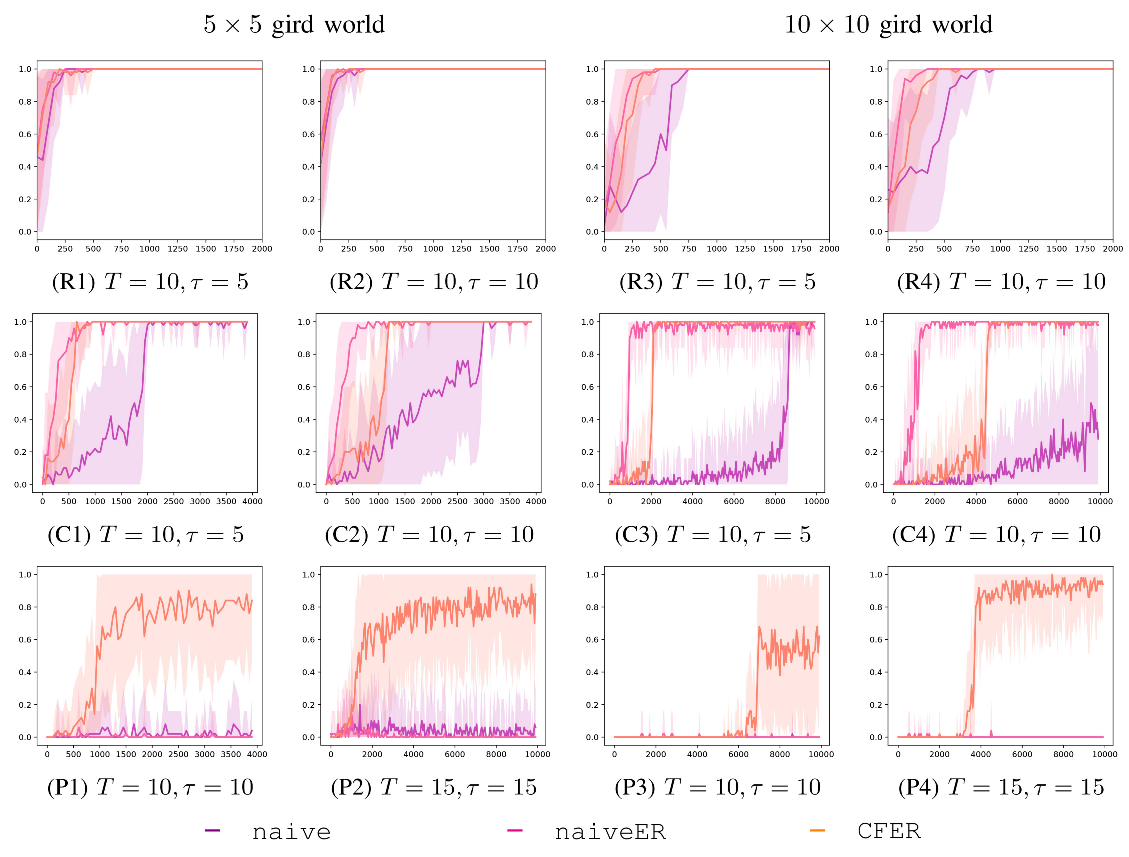

Fig. 3. Satisfaction curves for `naive`, `naiveER`,`CFER` for reaching, charging and patrolling task during training.

Reaching task: The agent is supposed to reach the storage area (the blue colored region in Figure 2) to report to duty in $T + \tau$ time units. Note that, although our fragment Φ requires two nested temporal operator, this task can still be captured by

$$\Phi_1 = \mathbf{F}_{[0,T+\tau)}(s \in A) \ = \mathbf{F}_{[0,T)}\mathbf{F}_{[0,\tau)}(s \in A). \tag{15}$$

Charging task: The robot needs to visit the charging station every T time units and for each visit, to stay inside the charging region for certain time τ to get fully charged. This task can be specified by

$$\Phi_2 = \mathbf{F}_{[0,T)}\mathbf{G}_{[0,\tau)}(s \in \text{Charging Region}). \tag{16}$$

Patrolling task: During the working hour, the robot must pick up parcels from one area and deliver them to another, completing each cycle within τ time units by patrolling between blue and red regions in Figure 2. This task is specified as

$$\Phi_3 = \mathbf{G}_{[0,T)}\big[\mathbf{F}_{[0,\tau)}(s \in \text{A}) \wedge \mathbf{F}_{[0,\tau)}(s \in \text{B})\big]. \tag{17}$$

B. Implementation Details of Learning Algorithm

We use the standard tabular Q-learning algorithm in [15] as a baseline to synthesize control policies and compare it to our modified version denoted as `CFER`. We adopt a uniform sampling strategy to sample from the buffer. Furthermore, to investigate the effect of counterfactual experience generation and experience replay, we also conduct tests with ordinary experience replay, which we denote as `naiveER`. Specifically, in `naiveER`, only factual experiences are inserted into the replay memory in contrast to `CFER`, where counterfactual experiences are also used for replay.

TABLE I
IMPLEMENTATION PARAMETERS

index	n_d	memory size	replay period	batch size	scale function	max episodes
R1-R4	400	1000	4	64	$2r-1$	4000
C1-C2	400	1000	4	64	$2r-1$	4000
C3-C4	400	1000	4	64	$2r-1$	10000
P1	400	1000	4	64	$r-0.9$	4000
P2-P4	1000	10000	4	256	$r-0.9$	10000

Note that for each of the three tasks, we consider two different pairs of time horizons T and τ. Recall that we consider two different resolutions of abstraction: 5×5 and 10×10. Therefore, we essentially run the three learning algorithms on $3 \times 2 \times 2 = 12$ different scenarios.

To make a fair comparison, each algorithm is implemented using the identical hyper-parameters as specified in Table I. Additionally, shared parameters we use throughout all experiments are $\beta = 50$ for the reward function approximation, learning rate $\alpha = 0.01$, and discount factor $\gamma = 0.9999$ for the policy update(5). We decay the exploration ratio ϵ by $\epsilon = \epsilon_\infty + (\epsilon_\infty - \epsilon_0) \times e^{-\text{episode}/n_d}$, where $\epsilon_0 = 0.99$ and $\epsilon_\infty = 0.01$ denotes the maximum and minimum value of ϵ and n_d, a parameter that determines the rate of the decay. During the learning process, the policy is evaluated every 20 episodes with the average success rates, the result of which is visualized in the learning curve in Figure 3. The success rate at the end of the curve demonstrates the test success rate after training.

C. Simulation Results and Discussions

We analyze our results from the following perspectives:

1) Is Each Task Achieved?: To demonstrate the feasibility of our algorithm, in Figure 2, we depict trajectories observed in the last evaluation with the maximum accumulative reward. Specifically, Figure 2(a)-(c) correspond to 5×5 grids with $\tau = 5$, and Figure 2(d)-(f) correspond to 10×10 grids with $\tau = 10$. In all of these six scenarios, the CFER policies are able to generate satisfying trajectories after a reasonable time of learning, thereby verifying the feasibility of the CFER algorithm. It is worth noting that, for the patrolling task, only CFER policies successfully generate trajectories that meet the task specification. The reason behind this observation will be analyzed later along with the learning curve.

2) Does CFER Facilitate Learning?: Figure 3 depicts the learning curves for the entire 12 sets of experiments with three different algorithms. It is commonly observed that in each sub-figure, the CFER curve converges much faster than the naive one. However, in the reaching and charging task, the naiveER policy converges sightly faster. But in the patrolling task, only CFER policy manages to converge to a non-zero success probability within given training episodes.

Our explanation is that the efficiency of CFER compared with naiveER will become more significant when the complexity of task increases. For $\Phi_1 = \mathbf{F}_{[0,10)}\mathbf{F}_{[0,5)}(s \in A)$ in a 10×10 grid world, the size of the state space is $5 \times 10^2 = 500$, which is smaller than the replay memory size 1000. Since it is easy to visit all the 500 states, having factual transitions filled into the replay memory already ensures the variety of exploration. However, due to the limited replay memory size, the generation of *counterfactual* experience homogenizes the replay memory by producing repetitive transitions across, leading to a less ideal learning curve. However, for the patrolling task, where substantial improvement of CFER is observed across all experiments, the sizes of the state spaces are $2500, 5625, 10000$, and 22500, and replay memory is set to $10,000$. While it is burdensome for the agent to explore the state space on its own or even fill up the replay memory, the generation of *counterfactual* transitions saves it from unnecessary trials and errors.

3) Does CFER Improve Scalability?: It is worth noting that the task difficulty increases from the reaching task to the charging task and patrolling task. Additionally, task difficulty escalates with increasing horizons. Figure 3 demonstrates that the gap between CFER's learning curve and the naive learning curve widens as task difficulties increase. For tasks like patrolling, which cannot be completed within a finite time using naive or naiveER methods, CFER can effortlessly achieve the desired results. Moreover, since experience generations are performed on CPUs, the computation time is negligible compared to the time consumption of a single interaction. This provides evidence that our algorithm will considerably expedite learning on large-scale complex tasks described by STL formulae.

VI. Conclusion

In this letter, we investigate the control synthesis problem for STL tasks. We observe in experiments that the CFER policies generally converge faster compared to the naive policies. Specifically, our CFER approach proves particularly effective for handling complex STL tasks featuring long horizons and multiple sub-tasks in large state spaces. A potential extension of this letter is to achieve a balance between the roles of *factual* and *counterfactual* experience replay. An effective sampling scheme can be explored to unleash the potential of CFER.

References

[1] M. Hekmatnejad et al., "Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic," in *Proc. ACM-IEEE Int. Conf. Formal Methods Models Syst. Design*, 2019, pp. 1–11.

[2] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," in *Proc. 16th Int. Conf. Hybrid Syst. Comput. Control*, 2013, pp. 43–52.

[3] G. Silano, T. Baca, R. Penicka, D. Liuzza, and M. Saska, "Power line inspection tasks with multi-aerial robot systems via signal temporal logic specifications," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 4169–4176, Apr. 2021.

[4] R. Yan and A. A. Julius, "Interpretable seizure detection with signal temporal logic neural network," *Biom. Signal Process. Control*, vol. 78, Sep. 2022, Art. no 103998.

[5] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *Proc. IEEE Conf. Decis. Control*, 2014, pp. 81–87.

[6] V. Kurtz and H. Lin, "Mixed-integer programming for signal temporal logic with fewer binary variables," *IEEE Control Syst. Lett.*, vol. 6, pp. 2635–2640, 2022.

[7] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 3451–3458, Apr. 2022.

[8] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for signal temporal logic tasks," *IEEE Control Syst. Lett.*, vol. 3, pp. 96–101, 2019.

[9] L. Lindemann and D. V. Dimarogonas, "Barrier function based collaborative control of multiple robots under signal temporal logic tasks," *IEEE Trans. Control Netw. Syst.*, vol. 7, no. 4, pp. 1916–1928, Dec. 2020.

[10] W. Xiao, C. A. Belta, and C. G. Cassandras, "High order control Lyapunov-barrier functions for temporal logic specifications," in *Proc. Am. Control Conf.*, 2021, pp. 4886–4891.

[11] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "*Q*-learning for robust satisfaction of signal temporal logic specifications," in *Proc. IEEE 55th Conf. Decis. Control*, 2016, pp. 6565–6570.

[12] D. Muniraj, K. G. Vamvoudakis, and M. Farhood, "Enforcing signal temporal logic specifications in multi-agent adversarial environments: A deep q-learning approach," in *Proc. IEEE Conf. Decis. Control*, 2018, pp. 4141–4146.

[13] A. Balakrishnan and J. V. Deshmukh, "Structured reward shaping using signal temporal logic specifications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2019, pp. 3481–3486.

[14] K. C. Kalagarla, R. Jain, and P. Nuzzo, "Model-free reinforcement learning for optimal control of Markov decision processes under signal temporal logic specifications," in *Proc. 60th IEEE Conf. Decis. Control*, 2021, pp. 2252–2257.

[15] H. Venkataraman, D. Aksaray, and P. Seiler, "Tractable reinforcement learning of signal temporal logic objectives," in *Proc. Learn. Dyn. Control*, 2020, pp. 308–317.

[16] J. Ikemoto and T. Ushio, "Deep reinforcement learning under signal temporal logic constraints using lagrangian relaxation," *IEEE Access*, vol. 10, pp. 114814–114828, 2022.

[17] N. K. Singh and I. Saha, "STL-based synthesis of feedback controllers using reinforcement learning," in *Proc. 37th AAAI Conf. Artif. Intell.*, 2023, pp. 15118–15126.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT press, 2018.

[19] C. Voloshin, A. Verma, and Y. Yue, "Eventual discounting temporal logic counterfactual experience replay," 2023, *arXiv:2303.02135*,

[20] D. Shao and M. Kwiatkowska, "Sample efficient model-free reinforcement learning from LTL specifications with optimality guarantees," 2023, *arXiv:2305.01381*.