Temporal Logic Guided Meta Q-Learning of Multiple Tasks

Hao Zhang and Zhen Kan, *Senior Member, IEEE*

***Abstract*—Reinforcement learning (RL) based approaches have enabled robots to perform various tasks. However, most existing RL algorithms focus on learning a particular task, without considering generalization to new tasks. To address this issue, by combining meta learning and reinforcement learning, we develop a meta Q-learning of multi-task (MQMT) framework where the robot effectively learns a meta model from a diverse set of training tasks and then generalizes the learned model to a new set of tasks that have never been encountered during training using only a small amount of additional data. Particularly, the multiple tasks are specified by co-safe linear temporal logic specification. As a semantics-preserving rewriting operation, LTL progression is exploited to decompose training tasks into learnable sub-goals, which not only enables simultaneous learning of multiple tasks, but also facilitates reward design by converting non-Markovian reward process to Markovian ones. Reward shaping is further incorporated into the reward design to relax the sparse reward issue. The simulation and experiment results demonstrate the effectiveness of the MQMT framework.**

***Index Terms*—Meta Q-learning, multiple tasks, linear temporal logic.**

I. Introduction

ONE of the ultimate goals in robotic learning is to let the robot evolve continuously like humans via interactions with the environment. To enable such human-level intelligence, the capability of learning multiple tasks and generalizing the learned skills to new tasks or new environments that have never been encountered during training is crucial. Among various learning algorithms, reinforcement learning (RL) is a sequential decision-making process [1]. Although RL based approaches have enabled robots to perform tasks from simple to complex ones (e.g., AlphaGo and Atari games), an more important yet challenging topic is how the robot can learn multiple tasks and efficiently extend the learned skills to new tasks with limited exposure to the new configurations. In particular, there are three main challenges: 1) unlike many RL algorithms that focus on learning a particular task, how can the robot efficiently learn multiple tasks at the same time? 2) Since many practical tasks require the robot to perform a series of logically organized sub-tasks (e.g., trigger the alarm, find the extinguisher, and then put out the fire), resulting in non-Markovian reward decision process (NMRDP), how can the non-Markovian rewards be properly handled to facilitate learning? 3) When encountering new tasks that have never been seen during training, how can the robot extend the learned skills to new tasks?

Recently, growing research has been devoted to meta reinforcement learning (meta-RL), which aims at solving unseen tasks fast and efficiently after trained over a distribution of relevant tasks. The idea of meta-RL was originally presented in [2] and [3], in which a model with memory is built to train the agent with a variety of tasks and environments. In [4], a model-agnostic meta-learning framework is developed, which is compatible with any model trained with gradient descent and can produce improved generalization performance on new tasks. The work of [5] presents a gradient-based meta-learning algorithm to optimize the return, which is a sampled and bootstrapped approximation of the true value function. Evolutionary strategies are incorporated in [6] to improve the agent adaptation ability with evolved policy gradient. Different from on-policy updating, a probabilistic off-policy meta-RL with latent context variables is developed in [7], which enhances adaptation efficiency using posterior sampling during training. Based on off-policy methods, a maximum average rewards approach is developed in [8], which reuses data from the training phase to improve sampling efficiency and leverages deterministic context variables to improve the adaptation to unseen tasks. Despite recent progress, meta-RL still requires intrinsic connections among training tasks, which imposes significant challenges to achieve good generalization to unseen tasks. The lack of valid or sufficient data of the current task during the adaptation phase imposes another challenge to meta-RL. In addition, most of the existing meta-RL methods focus on single-goal tasks. It is unclear how conventional meta-RL can be extended to handle multi-goal tasks that consist of a series of logically organized sub-tasks.

Linear temporal logic (LTL), as a formal language, is capable of describing a wide range of complex tasks composed of logically organized sub-tasks [9]–[11]. Recently, temporal logic is increasingly being used with learning algorithms to facilitate motion planning of robotic systems. For instance, modular deep reinforcement learning is incorporated with temporal logic to enable continuous motion planning of an autonomous dynamical system in [12]. Learning-based probabilistic motion planing subject to temporal logic constraints in the presence of environment and motion uncertainties is investigated in [13]. Truncated linear temporal logic is leveraged to facilitate the reward design in [14], which improves the performance of reinforcement learning in robotic planning. However, most of these aforementioned results focus on learning a particular

Manuscript received 23 February 2022; accepted 12 June 2022. Date of publication 22 June 2022; date of current version 7 July 2022. This letter was recommended for publication by Associate Editor M. Khoramshahi and Editor J. Kober upon evaluation of the reviewers' comments. This work was supported the National Natural Science Foundation of China under Grant U2013601 and 62173314. *(Corresponding author: Zhen Kan.)*

The authors are with the Department of Automation, University of Science and Technology of China, Hefei 230026, China (e-mail: zcharlie0257@mail.ustc.edu.cn; zkan@ustc.edu.cn).

Digital Object Identifier 10.1109/LRA.2022.3185384

task, resulting in poor generalization to unseen tasks. When considering multi-task learning, the work of [15] uses linear temporal logic to specify a set of tasks in a manner that supports the composition of learned skills and then develops an off-policy RL to speed up multi-task learning. Although the method in [15] can learn multi-task efficiently, it still lacks generalization to unseen tasks. To improve the adaptation of learned skill to new tasks, the work of [16] specifies the unseen task as an LTL formula and uses a compositional recurrent neural network as an encoder to train the learning agent to understand LTL semantics of the task. However, the reward for sub-task completion has to be manually designed in [16]. In [17], a neuro-symbolic agent is developed which combines deep reinforcement learning with temporal logic to achieve systematic generalization to unseen tasks. However, the approach is myopic, since it only optimizes for solving the next sub-task without considering what the agent must do after. In [18], the compositional syntax and the semantics of LTL are exploited to enable adaptation to new tasks.

Contributions: In this work, by combining meta learning and reinforcement learning, we develop a meta Q-learning of multiple tasks (MQMT) framework to effectively learn a meta model from multiple training tasks, such that the learned model can be efficiently generalized to new tasks or new environments that have never been encountered during training using only a small amount of additional data. Particularly, the multiple tasks are specified by co-safe LTL. As a semantics preserving rewriting operation, LTL progression is exploited to decompose training tasks into learnable sub-goals, which not only enables simultaneous learning of multiple tasks to significantly improve the learning efficiency, but also facilitates the reward design by converting non-Markovian reward process to Markovian ones. Inspired by [19] and [20], reward shaping is further incorporated into the reward design to relax the sparse reward issue. The simulation and experiment results demonstrate the effectiveness of the MQMT framework.

II. Preliminaries

A. Co-Safe Liner Temporal Logic

Linear temporal logic is a formal language widely used to describe complex (possibly infinite-length) tasks. Co-safe LTL (sc-LTL) is a subclass of LTL that can be fulfilled by finite-length state trajectories [21]. Since sc-LTL is suitable for describing robotic tasks, e.g., delivery and pick-and-place, this work focuses on sc-LTL. An sc-LTL formula is built on a set of atomic propositions Π that can be true or false, standard Boolean operators such as $\wedge$ (conjunction), $\vee$ (disjunction), and $\neg$ (negation), and temporal operators such as $\bigcirc$ (next), $\diamondsuit$ (eventually), and $\cup$ (until). The semantics of an sc-LTL formula are interpreted over a word $\sigma = \sigma_0\sigma_1 \ldots \sigma_n$, which is a finite sequence with $\sigma_i \in 2^{\Pi}$, $i = 0, \ldots, n$, where 2^{Π} represents the power set of Π. Denote by $\langle \sigma, i \rangle \vDash \varphi$ if the sc-LTL formula φ holds from the position i of σ. More detailed treatment can be found in [10].

B. Meta-Reinforcement Learning

Meta-RL generally consists of a training phase and an adaptation phase. The training phase aims at training a robot over a distribution of training tasks, so that in the adaptation phase the trained robot, even with limited exposure to the new configurations, can learn new skills or adapt to new tasks fast and efficiently. In this work, we consider a group of training tasks $\Phi = \{\varphi_i\}_{i=1,..,n}$, which are specified by sc-LTL formulas. When performing the task $\varphi_i \in \Phi$, the interaction of the robot with the environment is modeled by a discrete labeled MDP $\mathcal{M}_i = (S, T_i, A, p_i, \Pi, L, R_i, \gamma, \mu_i)$, where S is a finite set of states, $T_i \subseteq S$ is a finite set of terminal states, A is a finite set of actions, $p_i(s'|s,a)$ is the transition probability from $s \in S$ to $s' \in S$ under action $a \in A$, Π is a set of atomic propositions indicating the properties associated with the states, $L : S \to 2^{\Pi}$ is the labeling function, R_i is the reward function, $\gamma \in (0, 1]$ is the discount factor, and μ_i is the initial state distribution. The labelling function L can be considered as a collection of event detectors that fire when $p \in \Pi$ holds in the environment, which enables the robot to evaluate whether or not an LTL specification is satisfied. It is assumed that the transition probability p_i is unknown and all training tasks in Φ share the same workspace S and action space A, but with different terminal states T_i, initial state distribution μ_i, and reward function R_i.

For each task $\varphi_i \in \Phi$, the robot interacts with the environment following a policy π_i over $\mathcal{M}_i$. Specifically, the robot starts from an initial state s_0 sampled from μ_i in each episode, and transits from the current state s_t to the next state s_{t+1} sampled from $p_i(s_{t+1}|s_t, a_t)$ under the control action a_t generated by the policy π_i. The robot then receives a reward r_t. The Q-value is $Q_i(s,a) = \mathbb{E}[r_0 + \gamma r_1 + \cdots | s_0 = s, a_0 = a, \pi_i]$ and the optimal Q-value is $Q_i^*(s,a) = \max_{\pi_i} Q_i(s,a)$. The optimal policy π_i^* can then be derived from the optimal Q-value [1]. If a large state space is considered, function approximation is often used on the Q-value function. For instance, Deep Q-Networks (DQN) uses deep neural networks for Q-value function approximation [22], i.e., $Q(s,a;\theta)$ with weights θ. The conventional reward function is often assumed to be Markovian, i.e., the reward received at s_{t+1} only depends on the transition from s_t to s_{t+1}. However, in practice, the robot is often rewarded based on the completion of the assigned task φ_i, i.e., $\sigma \vDash \varphi_i$, and the episode ends as soon as φ_i is satisfied or falsified. Since the word $\sigma = \sigma_0\sigma_1 \ldots \sigma_t$ is defined over the state trajectory $s_0 s_1 \ldots s_t$ via the labeling function L, in this work we consider non-Markovian reward function $R_i(s_0 s_1 \ldots s_t)$. We will discuss how the challenge of non-Markovian rewards can be handled in the sequel.

Given the set $\{\mathcal{M}_i\}_{i=1,..,n}$ corresponding to all tasks in Φ, the goal of the training phase is to learn a meta-model with weights θ_{meta} that maximizes the average rewards across all training tasks, i.e.,

$$\theta_{\text{meta}} = \underset{\theta}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} l_{\text{meta}}^{i}(\theta) \tag{1}$$

where $l_{\text{meta}}^{i}(\theta)$ is a meta-training loss dependent on particular meta-update methods [8]. In the adaptation phase, the robot will first initialize the model weights as θ_{meta} and then adapt to new tasks by further updating the weights according to $\hat{\theta}_{\text{meta}} = \underset{\theta}{\text{argmax}} G_\theta(s)$, where $G_\theta(s) = \mathbb{E}^{\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right]$ is the expected discounted return for new tasks.

III. Problem Formulation

To elaborate the proposed multi-task meta-learning algorithm, the following example will be used as a running example throughout the work.

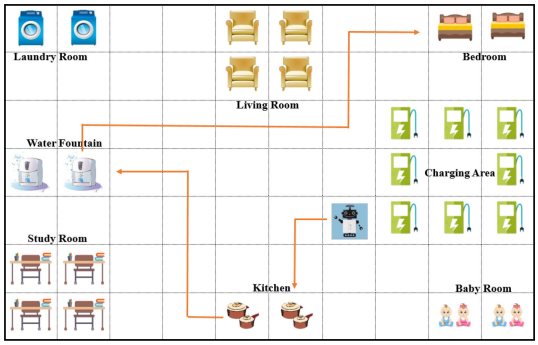

Fig. 1. Example of a robot offering a variety of home services (e.g., $\varphi_{\texttt{water}}$) by sequentially visiting a set of locations.

Example 1: Consider a grid home environment as shown in Fig. 1, in which the robot can offer a variety of home services by sequentially visiting a set of locations. The set of propositions Π is {`Kitchen`, `Water_Fountain`, `Dinning_Room`, `Bedroom`, `Living_Room`, `Charge`, `Laundry`, `Baby_Room`}. Using above propositions in Π, an example home service can be formulated by an sc-LTL formula as $\varphi_{\texttt{water}} = \Diamond(\texttt{Kitchen} \wedge \Diamond(\texttt{Water\_Fountain} \wedge \Diamond\texttt{Bedroom}))$, which requires the robot to sequentially visit the kitchen (e.g., get a cup), the water fountain (e.g., fill the cup), and the Bedroom (e.g., deliver water). The robot trajectory is shown in orange arrow lines in Fig. 1. Besides delivering water, the robot can also perform various other tasks by combing temporal and Boolean operators over the propositions in Π. For instance, $\varphi_{\texttt{care}} = \Diamond(\texttt{Baby\_Room} \wedge \Diamond\texttt{Charge})$ requires the robot to visit the baby room first and then get charged in the charging area, and $\varphi_{\texttt{clean}} = \Diamond(\texttt{Bedroom} \wedge \Diamond\texttt{Charge})$ requires the robot to visit the bedroom before getting charged.

In this work, unlike many existing works that focus on learning a particular task, we are interested in meta-learning of multiple tasks. That is, we are concerned about learning a model on a set of training tasks such that the learned model can efficiently adapt to new and unseen tasks using only a small amount of new data. For instance, by learning from tasks such as $\varphi_{\texttt{water}}$, $\varphi_{\texttt{care}}$, and $\varphi_{\texttt{clean}}$, we hope the robot can learn fast when encountering a new task such as $\varphi_{\texttt{trash}} = \Diamond(\texttt{Kitchen} \wedge \Diamond(\texttt{Bedroom} \wedge \Diamond\texttt{Charge}))$ to collect trash by sequentially visiting `Kitchen` and `Bedroom` before getting charged.

Specifically, suppose each training task φ_i is associated with the Q-value function $Q_i(s, a; \theta)$ approximated by a neural network parameterized with weights θ, where θ is updated according to meta-training loss $l^i_{\text{meta}}(\theta)$. The goal of meta-learning in this work is to find a meta-weight θ_{meta} over all tasks $\Phi = \{\varphi_i\}_{i=1,..,n}$, such that a small number of additional new samples can lead to fast learning on a new task. To this end, the problem can be formally presented as follows.

Problem 1: Given a finite (but potentially large) set of training tasks $\Phi = \{\varphi_i\}_{i=1,..,n}$ specified by sc-LTL formulas, a probability distribution τ over the training tasks in Φ, and a set of MDPs $\mathcal{M}_i = (S, T_i, A, p_i, \Pi, L, \gamma, \mu_i)$ corresponding to task φ_i with the reward function $R_{\varphi_i}(s_0 s_1 \dots s_t)$ to be designed, the goal of this work is to learn a model parameterized with the meta weight $\theta_{\text{meta}} = \underset{\theta}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} l^i_{\text{meta}}(\theta)$, where $l^i_{\text{meta}}(\theta)$ is a meta-training loss of task φ_i, so that the learned model parameterized with θ_{meta} can adapt to new and unseen tasks efficiently.

IV. Algorithm Design

In this section, we present a novel framework, namely meta Q-learning of multiple tasks (MQMT), to address Problem 1. Section IV-A presents how LTL progression can be incorporated in the design of the reward function to facilitate meta-learning. Section IV-B and Section IV-C explains in detail the training phase and the adaptation phase of MQMT.

A. LTL Progression and LTL-Guided Multi-Policy MDP

A major challenge in solving Problem 1 is that the reward function $R_{\varphi_i}(s_0 s_1 \dots s_t)$ used in the Q-value function depends on the history of states and thus is non-Markovian. To address the issue of non-Markovian rewards, recent works of [16] and [17] either encode the policy using a recurrent neural network or develop a reasoning module to facilitate task completion. However, the methods in [16] and [17] generally suffer from high computational cost and can be suboptimal. In this work, the LTL progression from [15] and [23] is exploited to overcome the non-Markovian issue.

Let $\text{AT}(\varphi)$ denote the proposition required to progress the current LTL formula. The LTL progression is defined as follows.

Definition 1: Give an LTL formula φ and a word $\sigma = \sigma_0 \sigma_1 \dots$, the LTL progression $\text{prog}(\sigma_i, \varphi)$ at step i, $\forall i = 0, 1, \dots$, is defined as follows:

$$
\begin{aligned}
\text{prog}\,(\sigma_i, p) &= \text{true if } p \in \sigma_i, \text{ where } p \in \Pi, \\
\text{prog}\,(\sigma_i, p) &= \text{false if } p \notin \sigma_i, \text{ where } p \in \Pi, \\
\text{prog}\,(\sigma_i, \neg\varphi) &= \neg\,\text{prog}\,(\sigma_i, \varphi)\,, \\
\text{prog}\,(\sigma_i, \varphi_1 \wedge \varphi_2) &= \text{prog}\,(\sigma_i, \varphi_1) \wedge \text{prog}\,(\sigma_i, \varphi_2)\,, \\
\text{prog}\,(\sigma_i, \varphi_1 \vee \varphi_2) &= \text{prog}\,(\sigma_i, \varphi_1) \vee \text{prog}\,(\sigma_i, \varphi_2)\,, \\
\text{prog}\,(\sigma_i, \circledcirc\varphi) &= \varphi, \\
\text{prog}\,(\sigma_i, \varphi_1 \cup \varphi_2) &= \text{prog}\,(\sigma_i, \varphi_2) \\
&\quad \vee (\text{prog}\,(\sigma_i, \varphi_1) \wedge \varphi_1 \cup \varphi_2)\,,
\end{aligned}
$$

$$
\text{prog}\,(\sigma_i, \varphi) = \begin{cases} \varphi \setminus p, & \text{if } \text{AT}(\varphi) = p, \text{prog}\,(\sigma_i, p) = \text{true}, \\ \varphi, & \text{otherwise.} \end{cases}
$$

At each step, the operator prog in Def. 1 takes as input an LTL formula φ and the current label σ_i, and outputs a formula to reflect which parts of the original specifications remain to be addressed. For example, if given a formula $\varphi = \Diamond(a \wedge \Diamond b)$ and a label a, the operator prog will progress φ to $\varphi' = \Diamond b$ otherwise it will not be progressed.

Theorem 1: [15] Given any LTL formula φ and the corresponding word $\sigma = \sigma_i \sigma_{i+1} \dots$, $\langle \sigma, i \rangle \vDash \varphi$ if and only if $\langle \sigma, i+1 \rangle \vDash \text{prog}(\sigma_i, \varphi)$.

The LTL progression in Def. 1 and Thm. 1 has the following benefits. First, since the operator prog is semantics-preserving, by iteratively applying prog after each step, the diminished remaining instructions indicate progression towards task completion. For instance, consider the task $\varphi_{\texttt{water}}$ in Example 1, which will progress to the subsequent sub-task $\Diamond(\texttt{Water\_Fountain} \wedge \Diamond\texttt{Bedroom})$ as soon as the robot reaches Kitchen. Since it can indicate the progress of tasks, it can be exploited to design a reward functions to reflect which partial task has been completed and which part has not yet. Another benefit is that, by iteratively using prog, the training tasks can be decomposed into a set of learnable sub-tasks to facilitate

simultaneous learning. For instance, applying the operator prog over the set of tasks $\{\varphi_{\texttt{care}}, \varphi_{\texttt{clean}}\}$, $\varphi_{\texttt{charge}} = \Diamond\texttt{charge}$ can be extracted as a sub-task since both $\varphi_{\texttt{care}}$ and $\varphi_{\texttt{clean}}$ can be progressed to $\varphi_{\texttt{charge}}$. Hence, the tasks $\{\varphi_{\texttt{care}}, \varphi_{\texttt{clean}}, \varphi_{\texttt{charge}}\}$ will be viewed as an extended training set to train the meta weightsθ_{meta}. In the subsequent development, we denote by Ψ_i the extended training set for $\varphi_i \in \Phi$, which consists of φ_i and its progressed sub-tasks.

Besides using LTL progress, we further incorporate Z from [8] to capture the history knowledge and avoid perceptual aliasing in discrete environments. Specifically, leveraging the structure in [24] that contains a recurrent layer without convolutional layers, the GRU model from [25] is used as the recurrent layer due to the trade-off between its powerful functionality and computational complexity. The Q-value function $Q(s, a; \theta)$ is then conditioned on the context as $Q(s, a, z; \theta)$, where $z \in Z$ is a deterministic context variable.

Based on the LTL progression in Def. 1 and the context variable, an augmented MDP, namely LTL-guided multi-policy MDP (LMMDP), is developed as follows.

Definition 2: Given a discrete labeled MDP $\mathcal{M}_i = (S, T_i, A, p_i, \Pi, L, \gamma, \mu_i)$ corresponding to an LTL task $\varphi_i \in \Phi$, the LMMDP is constructed by augmenting $\mathcal{M}_i$ to $\mathcal{M}_{\Psi_i} \triangleq \{(\tilde{S}, \tilde{T}_i, A, \tilde{p}_i, \Pi, L, \tilde{R}_{\phi_j}, \gamma, \tilde{\mu}) : \phi_j \in \Psi_i, j = 1, \ldots, |\Psi_i|\}$ with $|\Psi_i|$ indicating the number of tasks in Ψ_i, where $\tilde{S} = S \times \Psi_i$, $\tilde{T}_i = \{(s, \phi_j) | s \in T_i \text{or} \phi_j \in \{\text{true}, \text{false}\}, \phi_j \in \Psi_i\}$, $\tilde{p}_i((s', \phi'_j)|(s, \phi_j), a) = p_i(s'|s, a)$ if $\phi'_j = \mathrm{prog}(L(s), \phi_j)$ and $\tilde{p}_i((s', \phi'_j)|(s, \phi_j), a) = 0$ otherwise, $\tilde{\mu}(s, \varphi) = \mu_i(s) \cdot \tau(\varphi_i)$ with $\tau(\varphi_i)$ indicating the probability of selecting φ_i from Φ, and $\tilde{R}_{\phi_j}$ is the reward function associated with the task $\phi_j \in \Psi_i$.

To design the reward function $\tilde{R}_{\phi_j}$ in Def. 2, we first design a base reward function as

$$r_{\varphi_i}\left(\phi_j, a, \phi'_j\right) = \begin{cases} 1, & \text{if } \phi'_j = \text{true}, \\ 0, & \text{otherwise}, \end{cases}$$

which indicates that the robot only receives a reward of 1 when the current task φ_i completes, as $\phi'_j = \text{true}$ indicates the completion of φ_i. Since the base reward function r_{φ_i} is always zero before φ_i completes, the reward might become sparse. To address this issue, inspired by [19] and [20], we design the reward function $\tilde{R}_{\phi_j}(\phi_j, a, \phi'_j)$ by reshaping r_{φ_i} using a potential function $\Omega : \Psi_i \to \mathbb{R}_{\leq 0}$ as

$$\tilde{R}_{\phi_j} = \begin{cases} r_{\varphi_i}\left(\phi_j, a, \phi'_j\right) \\ \quad + \gamma\Omega\left(\phi'_j\right) - \Omega\left(\phi_j\right), & \text{if } \mathrm{prog}\left(L(s), \phi_j\right) = \phi'_j, \\ \quad (\gamma - 1)\Omega\left(\phi_j\right), & \text{otherwise}, \end{cases} \tag{2}$$

where $\gamma < 1$, $\Omega(\phi_j) = -v^*(\phi_j)$, where $v^*(\phi_j) = \max_a Q(\phi_j, a)$ is the value function, and $\Omega(\phi_j) = 0$ if $\phi_j = \text{true}$. The value function $v^*(\phi_j)$ can be calculated by value iteration in Alg. 1. As shown in [26], given any MDP model, transforming the reward function using a potential function Ω as in $\tilde{R}_{\phi_j}$ will not change the set of optimal policies. Thus, the learning performance can be improved by incorporating the real-valued potential function Ω while guaranteeing that the resulted policies via $\tilde{R}_{\phi_j}$ are still optimal with respect to the base reward function r_{φ_i}. Although conceptually the reward shaping is similar to [19] and [20], the reward function in this work is shaped based on LTL progression.

Algorithm 1: Value Iteration for Reward Shaping.

procedure INPUT: the extend set Ψ_i, a, r_{φ_i}, γ, Π
  for $\phi_j \in \Psi_i$ **do**
    $v^*(\phi_j) \leftarrow 0$ %initializing the value function
  end for
  $e \leftarrow 1$
  while $e > 0$ **do**
    $e \leftarrow 0$
    for $\phi_j \in \Psi_i$ **do**
      $v' \leftarrow \max_a\{r_{\varphi_i}(\phi_j, a, \phi'_j) + \gamma v^*(\phi'_j)|$
        $\phi'_j = \mathrm{prog}(\sigma_i, \phi_j), \sigma_i \in 2^{\Pi}\}$,
      $e = \max\{e, |v^*(\phi_j) - v'|\}$
      $v^*(\phi_j) \leftarrow v'$
    end for
  end while
  return v^*
end procedure

B. Training Phase of MQMT

The idea of the training phase behind MQMT is to extract sub-tasks from Φ via LTL progression as described in Sec. IV-A and use Q-learning method to simultaneously learn these sub-tasks. The trained agents will then be used in the adaptation phase to enable rapid learning of unseen tasks. The overall method behind MQMT is illustrated in Fig. 2(b) and the pseudo-code is outlined in Alg. 2.

We first extract sub-tasks from Φ by LTL progression to generate an extended training set $\tilde{\Phi} = \cup_{i=1}^{n} \Psi_i$, where $|\tilde{\Phi}| = k$. All tasks $\tilde{\phi} \in \tilde{\Phi}$ are associated with a Q-value function $Q_{\tilde{\phi}}(s, a, z; \theta)$ parameterized with θ and MQMT performs a series of episodes over the tasks in $\tilde{\Phi}$ using Q-learning method. Specifically, in each episode, suppose a task $\tilde{\phi} \in \tilde{\Phi}$ is selected (e.g., using a curriculum learning). Given the current state s, let $\tilde{\phi}' = \mathrm{prog}(L(s), \tilde{\phi})$ be the progressed task. The robot selects an action a following the ε-greedy policy based on $Q_{\tilde{\phi}}$ and then transits to the next state with rewards received by (2).

The next step is to update the Q functions. Let $\mathcal{D}_{\mathrm{meta}} = \cup_{i=1}^{n} \mathcal{M}_{\Psi_i}$ denote the augmented meta-training set, where $\mathcal{M}_{\Psi_i}$ is defined in Def. 2. As revealed in [8], the training phase of conventional MAML in [4] is to update the weights θ_{meta} in (1) towards the loss spaces that under-fit all tasks $\tilde{\phi} \in \tilde{\Phi}$. Motivated by this observation, we design the MQMT with weights θ_{meta} updated according to $\mathcal{D}_{\mathrm{meta}}$ as

$$\theta_{\mathrm{meta}} = \underset{\theta}{\mathrm{argmin}} \frac{1}{k} \sum_{i=1}^{k} l^i_{\mathrm{meta}}\left(\theta\right), \tag{3}$$

where θ is the weight of the Q function associated with all $\mathcal{M}_{\Psi_i}$ and $l^i_{\mathrm{meta}}(\theta)$ is a meta-training loss for each $\tilde{\phi}_i \in \tilde{\Phi}$.

Conventional RL algorithm usually searches randomly during training process. If an action effective for other tasks rather than the current task is encountered, such an action in general is discarded and will not be used to update the Q-value of other tasks, leading to low sampling efficiency and delayed convergence to the optimal policy. Differently, MQMT focuses on simultaneous Q function updates for all tasks in $\tilde{\Phi}$. Particularly, for each $\tilde{\phi} \in \tilde{\Phi}$, the robot updates the Q functions as if

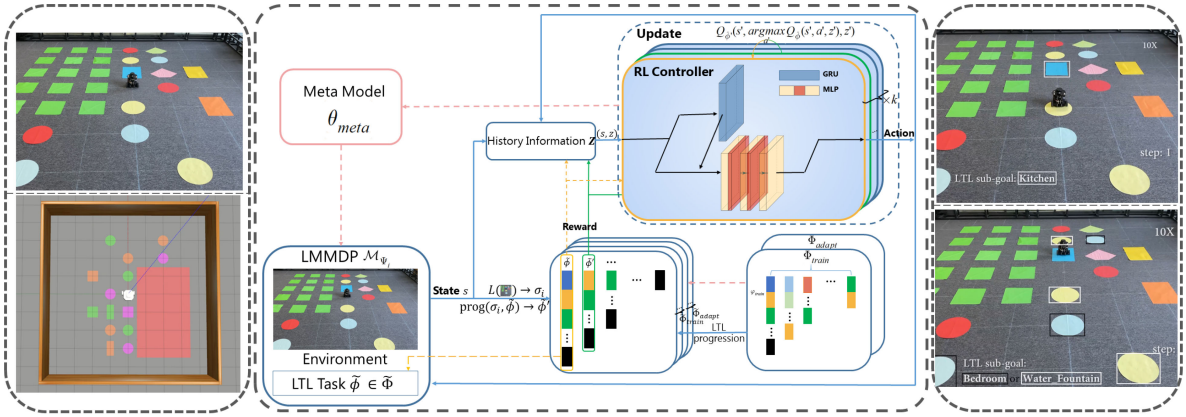

Fig. 2. (a) The experiment environments in the Gazebo/ROS simulation and real world. (b) The MQMT framework. In the training phase, the training tasks Φ are progressed to the extended tasks $\tilde{\Phi}$ first. Then during each episode, with the help of LTL progression, the Q-values of sub-tasks in $\tilde{\Phi}$ are learned simultaneously, which are then used to determine the meta weights θ_{meta}. In the adaptation phase, the meta weights θ_{meta} are further updated according to new tasks Φ_{adapt}. (c) The top and bottom figures represent the sequential and interleaving cases.

it is currently trying to solve $\tilde{\phi}$. Given the transitioned state s' and the progressed formula $\tilde{\phi}' = \text{prog}(L(s'), \tilde{\phi})$, the reward is determined by checking the resulting formula $\tilde{\phi}'$ using (2). Let $Q_{\tilde{\phi}}(s, a, z; \theta)$ and $Q_{\tilde{\phi}'}(s, a, z; \theta)$ be the Q function of task $\tilde{\phi}$ and $\tilde{\phi}'$, respectively. There exists two possible cases that $\tilde{\phi} = \tilde{\phi}'$. If $\tilde{\phi}'$ is neither true nor false and s' is not a dead-end of the environment, $Q_{\tilde{\phi}}$ is updated following a modified double DQN as

$$Q_{\tilde{\phi}} \leftarrow Q_{\tilde{\phi}} + \alpha \Big(\tilde{R}_{\tilde{\phi}} + \gamma Q_{\tilde{\phi}'} \Big(s,' \underset{a'}{\text{argmax}} Q_{\tilde{\phi}}(s,' a,' z'; \theta), z'; \theta^{-} \Big) - Q_{\tilde{\phi}} \Big)' \tag{4}$$

where θ^{-} are the target network weights of double DQN. By this way, the Q-value of $\tilde{\phi}$ will be propagated backwards from its sub-tasks $\tilde{\phi}'$. If s' is a terminal state or a dead-end when $\tilde{\phi}'$ is either true or false, (4) will be simplified as $Q_{\tilde{\phi}} \leftarrow Q_{\tilde{\phi}} + \alpha(\tilde{R}_{\tilde{\phi}} - Q_{\tilde{\phi}})$.

Thus by constructing LMMDP $\mathcal{M}_{\Psi_i}$, it will not only convert the non-Markovian reward process to Markovian ones, but also simultaneously update sub-task's Q-value. Such a method enables update of the current $Q_{\tilde{\phi}}$ using its sub-task $Q_{\tilde{\phi}'}$, leading to efficient learning of optimal policies.

Note that in the tabular case such a method always converges to the optimal policy. The optimality can be shown by proving that: 1) adding the context variable Z does not change the optimality of the original RL algorithm; 2) extracting sub-tasks from Φ via LTL progression and using Q-learning to simultaneously learn them can converge to an optimal policy for each task; and 3) the found policies by applying (2) are still optimal with respect to the original reward function. Since the points 2) and 3) have already been proved in [15] and [26] respectively, we focus on showing that adding the context variable does not change the optimality of the original RL algorithm.

Definition 3: Given a general MDP $\mathcal{M}_o = (S, T, A, p, R, \gamma, \mu)$, where S is a finite set of states, $T \subseteq S$ is a finite set of terminal states, A is a finite set of actions, $p(s'|s, a)$ is the transition probability from $s \in S$ to $s' \in S$ under action $a \in A$, R is the reward function, $\gamma \in (0, 1]$ is the discount factor, and μ is the initial state distribution. Then the extended MDP is constructed by augmenting $\mathcal{M}_o$ with the context variable Z to $\overline{\mathcal{M}}_o = (\overline{S}, T, A, \overline{p}, \overline{R}, \gamma, \mu)$, where $\overline{S} = S \times Z$, $\overline{p}((s', z') \mid (s, z), a)$ is the transition probability from $(s, z) \in \overline{S}$ to $(s', z') \in \overline{S}$ under action $a \in A$ and $\overline{R}((s, z), a, (s', z')) = R(s, a, s')$.

Theorem 2: If $\overline{\pi}_o^*$ is an optimal policy for $\overline{\mathcal{M}}_o$, then the policy $\overline{\pi}_o^*$ is also an optimal policy for $\mathcal{M}_o$.

Proof: Consider any policy π_o for $\mathcal{M}_o$. By definition in [1]:

$$Q_{\mathcal{M}}^{\pi_o}(s, a) = \mathbb{E}_{\pi_o, p} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid a_t = a \right]. \tag{5}$$

Since $\mathcal{M}_o$ and $\overline{\mathcal{M}}_o$ have the same A, T, R, γ, μ, we can construct an equivalent policy $\overline{\pi}_o$ for $\overline{\mathcal{M}}_o$, where $\pi_o(a|s) = \overline{\pi}_o(a|(s, z))$. Thus $p(s\´|s, a)$ is equivalent to $\overline{p}((s', z') \mid (s, z), a)$ for every $a \in A$. Replacing π_o by $\overline{\pi}_o$, p by $\overline{p}$, and R by $\overline{R}$ in (5) yields

$$Q_{\mathcal{M}}^{\pi_o}(s, a) = \mathbb{E}_{\overline{\pi}_o, \overline{p}} \left[\sum_{k=0}^{\infty} \gamma^k \overline{R}_{t+k+1} \mid a_t = a \right] = Q_{\overline{\mathcal{M}}}^{\overline{\pi}_o}(\overline{s}, a)$$

for any policy π_o, action a, and state s. In particular, if $\overline{\pi}_o^*$ is optimal in $\overline{\mathcal{M}}_o$, then $Q_{\mathcal{M}}^{\pi_o^*}(s, a) = Q_{\overline{\mathcal{M}}}^{\overline{\pi}_o^*}(\overline{s}, a) \geq Q_{\overline{\mathcal{M}}}^{\overline{\pi}_o}(\overline{s}, a) = Q_{\mathcal{M}}^{\pi_o}(s, a)$ for every policy $\overline{\pi}_o$ and action a. Therefore, $\overline{\pi}_o^*(a|(s, z)) = \pi_o^*(a|s)$ is optimal for $\mathcal{M}_o$. ■

C. Adaptation Phase of MQMT

Due to the use of double DQN [27], denote by $\mathscr{D}_{\text{meta}}$ the transition data generated in the training process and denote by $\mathscr{B}_{\text{train}}$ the experience replay buffer along with the context variables. Let $\mathscr{D}^{new}$ denote the transition data of new tasks $\tilde{\Phi}_{new}$. This section presents how the trained meta weights θ_{meta} from Sec. IV-B can be adapted to a new task.

Note that the direct use of transitions $\mathscr{D}_{\text{meta}}$ in $\mathscr{B}_{\text{train}}$ to update θ_{meta} will lead to at least two potential issues: 1) the transition data in $\mathscr{D}_{\text{meta}}$ in general is different from the data of new tasks in $\mathscr{D}^{new}$, and 2) even if the transitions are from

Algorithm 2: MQMT - Training and Adaptation Phases.

procedure Input: A finite set of LTL formulas $\Phi = \{\varphi_i\}_{i=1,..,n}$, a probability distribution τ over those formulas $\varphi_i \in \Phi$ and the MDP $\mathcal{M}_i$ corresponding to some φ
Output: The adapted meta weights $\hat{\theta}^*_{\text{meta}}$ for unseen tasks $\tilde{\phi}_{new} \in \tilde{\Phi}_{new}$
Initialization: The replay buffer $\mathscr{B}_{\text{train}}$, the task buffer $\mathscr{T}_{\text{train}}$, task case and map id for training
Training phase:
Extract sub-tasks as $\tilde{\Phi}$, initialize $Q_{\tilde{\phi}}$ and $Q_{\tilde{\phi}'}$ for $\tilde{\phi}$ and its sub-task $\tilde{\phi}'$
while $T < T_{max}$ **do**
Select a task $\tilde{\phi} \in \tilde{\Phi}$ as the current goal by a curriculum learning
while $t < t_{max}$ **do**
$\tilde{\phi}' \leftarrow \text{prog}(L(s), \tilde{\phi})$
if $\tilde{\phi}' \in \{\text{true,false}\}$ or $s \in T$ **then**
Break
end if
Gather data from $\tilde{\phi}$ and add it to $\mathscr{B}_{\text{train}}$ and $\mathscr{T}_{\text{train}}$
for $Q_{\tilde{\phi}} \in Q$ **do**
$\tilde{\phi}' \leftarrow \text{prog}(L(s), \tilde{\phi})$
Determine $\tilde{R}_{\tilde{\phi}}$ by (2) and update $Q_{\tilde{\phi}}$ following (4)
end for
$t \leftarrow t + 1$
end while
$T \leftarrow T + t$
end while
Return θ_{meta}, $\mathscr{B}_{\text{train}}$, and $\mathscr{T}_{\text{train}}$
Adaptation phase:
Initialize the adaptation buffer $\mathscr{B}_{adapt}$ for new LTL formulas $\tilde{\Phi}_{new}$
Extract sub-tasks as $\tilde{\Phi}_{new}$ and initialize the weights $\widehat{\theta}_{\text{meta}}$ as weights θ_{meta}
while $T < T_{max}$ **do**
Get task $\tilde{\phi}_{new}$ from $\tilde{\Phi}_{new}$ by a curriculum learning
while $t < t_{max}$ **do**
Repeat steps 6) - 9)
$\mathscr{B}_{adapt} \leftarrow$ Gather data from $\tilde{\phi}_{new}$ using $\widehat{\theta}_{\text{meta}}$
Fit $\beta(\tilde{\phi}_{new})$ using $\mathscr{B}_{adapt}$ and $\mathscr{B}_{\text{train}}$ and estimate $\widehat{ESS}$ by (7)
Repeat steps 11) - 14) and follow (6) in Sec. IV-C
$t \leftarrow t + 1$
end while
$T \leftarrow T + t$
end while
end procedure

the same task, using unseen states to update weights can lead to extrapolation error [28]. To address these issues, we will exploit propensity score [29] to characterize the data differences between $\mathscr{D}_{\text{meta}}$ and $\mathscr{D}^{new}$ and to re-weigh the odds to facilitate the update of the meta weight. Given the meta weight θ_{meta}, let $\widehat{\theta}_{\text{meta}}$ denote the updated weight for the adaptation to the new task $\mathcal{D}^{new}$. Specifically, given two data distributions $q(x)$ (e.g., training transitions $\mathscr{D}_{\text{meta}}$) and $p(x)$ (e.g., new transitions $\mathscr{D}^{new}$), the update law is designed as

$$\begin{aligned}\widehat{\theta}_{\text{meta}} \leftarrow \underset{\widehat{\theta}_{\text{meta}}}{\text{argmin}} \Bigg\{ & \mathop{\mathbb{E}}_{\zeta \sim \mathscr{D}^{new}} \left[l^a \left(\widehat{\theta}_{\text{meta}} \right) \right] \\ & + \mathop{\mathbb{E}}_{\zeta \sim \mathscr{D}_{\text{meta}}} \left[\beta \left(\zeta; \mathscr{D}^{new}, \mathscr{D}_{\text{meta}} \right) l^a \left(\widehat{\theta}_{\text{meta}} \right) \right] \\ & + (1 - \widehat{ESS}) \parallel \theta_{\text{meta}} - \widehat{\theta}_{\text{meta}} \parallel_2^2 \Bigg\}, \end{aligned} \tag{6}$$

where

$$\begin{aligned} l^a \left(\widehat{\theta}_{\text{meta}} \right) = \mathop{\mathbb{E}}_{s' \sim \zeta} \Bigg[\Bigg(& \tilde{R}_{\tilde{\phi}} + \gamma Q_{\tilde{\phi}'} \Bigg(s,' \underset{a'}{\text{argmax}} Q_{\tilde{\phi}}(s,' \\ & a,' z'; \widehat{\theta}_{\text{meta}}), z'; \widehat{\theta}_{\text{meta}} \Bigg) - Q_{\tilde{\phi}} \Bigg)^2 \Bigg] \end{aligned}$$

represents the TD error in the adaptation phase and

$$\widehat{\text{ESS}} = \frac{1}{m} \frac{(\sum_{k=1}^m \beta(x_k))^2}{\sum_{k=1}^m \beta(x_k)^2} \in [0, 1] \tag{7}$$

represents the normalized effective sample size. The term $\widehat{\text{ESS}}$ measures the difference between the distributions $q(x)$ and $p(x)$, where $\beta(\cdot)$ is the propensity score from [29]. We properly clip the value $\beta(\cdot)$ to prevent a higher variance in gradient descent update. A popular method to estimate $\widehat{\text{ESS}}$ is by Monte Carlo methods [30]. If two distributions $q(x)$ and $p(x)$ are close, $\widehat{\text{ESS}} \to 1$ and zero otherwise. In (6), the term

$$\underset{\widehat{\theta}_{\text{meta}}}{\text{argmin}} \left\{ \mathop{\mathbb{E}}_{\zeta \sim \mathscr{D}^{new}} \left[l^a \left(\widehat{\theta}_{\text{meta}} \right) \right] + \eta \parallel \theta_{\text{meta}} - \widehat{\theta}_{\text{meta}} \parallel_2^2 \right\},$$

with $\eta = \frac{1 - \widehat{ESS}}{2}$ aims at adapting to unseen tasks using $\mathscr{D}^{new}$, where $\widehat{\theta}_{\text{meta}}$ is initialized as θ_{meta}, $\parallel \theta_{\text{meta}} - \widehat{\theta}_{\text{meta}} \parallel$ is an adaptation proximal term which prevents degradation of the weights during update, and η is a hyper-parameter of quadratic penalty relaxation [31]. The remaining term

$$\begin{aligned} \underset{\widehat{\theta}_{\text{meta}}}{\text{argmin}} \Bigg\{ & \mathop{\mathbb{E}}_{\zeta \sim \mathscr{D}_{\text{meta}}} \left[\beta \left(\zeta; \mathscr{D}^{new}, \mathscr{D}_{\text{meta}} \right) l^a \left(\widehat{\theta}_{\text{meta}} \right) \right] \\ & + \eta \parallel \theta_{\text{meta}} - \widehat{\theta}_{\text{meta}} \parallel_2^2 \Bigg\} \end{aligned} \tag{8}$$

re-weighs the old data $\mathscr{D}_{\text{meta}}$ to update weights $\widehat{\theta}_{\text{meta}}$.

V. Case Studies

The performance of the MQMT framework is evaluated against the state-of-the-art algorithms in the following aspects. **1) Performance:** How well does MQMT outperform the leading algorithms in a discrete environment? **2) Architecture:** What is the role of context variable in MQMT? **3) Upward Generalization:** How well can the trained MQMT model be generalized to previously unseen tasks?

A. Simulation Results and Discussions

Consider a grid home-like environment as shown in Fig. 1 where the robot is assigned with sequential and interleaving tasks. In the training phase, the map size is 21×21. To show the adaptation capability to a different environment, a larger map of 25×25 is used in the adaptation phase in which the number and types of rooms remain the same as in the training map. In the

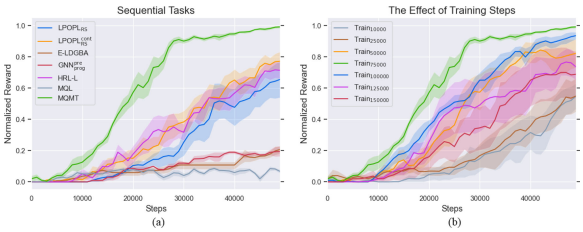

Fig. 3. (a) The performance of different algorithms for sequential tasks. (b) The performance of the adaptation phase under different training steps.

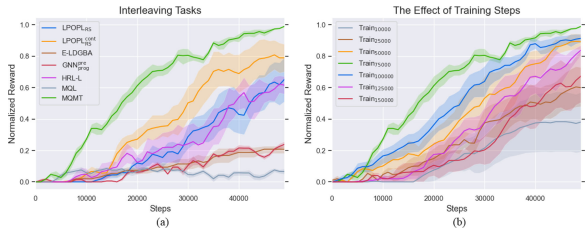

Fig. 4. (a) The performance of different algorithms for interleaving tasks. (b) The performance of the adaptation phase under different training steps.

adaptation phase, $\widehat{\theta}_{\text{meta}}$ are tested for all of the specified tasks after a learning period of 100 steps.

To show the effectiveness of the MQMT framework, it is empirically compared with six baselines. The first baseline is LPOPL_{RS}, which extends the LPOPL from [15] by incorporating reward shaping to speed up learning without compromising convergence guarantees in the grid world. The second baseline is $\text{LPOPL}_{\text{RS}}^{\text{cont}}$, which further incorporates LPOPL_{RS} with context variables. The third baseline is the E-LDGBA from [32] which integrates automaton-based RL algorithm with embedded LDGBA. The forth baseline is $\text{GNN}_{\text{prog}}^{\text{pre}}$, which is a method from [18] that exploits the compositional syntax and semantics of LTL by GNN to enable the adaptation to new tasks. To be a fair comparison, the GNN module is pre-trained in the same environment as in [18] and then applied to the downstream tasks (e.g., the adaptation tasks in the grid home-like environment). The fifth baseline is HRL-L, which incorporates the hierarchical RL algorithm from [33] with LTL pruning. The sixth baseline is MQL, which is the meta-RL algorithm from [8].

1) Case 1. Sequential Tasks: We first evaluate the MQMT framework for sequential tasks, which consists of a sequence of areas to be visited. For instance, the task of delivering an apple φ_{apple} is defined by sequentially visiting $Dinning\_Room$, Kitchen, and Living_Room, which can be written in an sc-LTL formula as $\varphi_{\text{apple}} = \Diamond(Dinning\_Room \wedge \Diamond(\text{ Kitchen} \wedge \Diamond \text{ Living\_Room}))$. More training and test sequential tasks can be found in the experiment video. We set $p_{succ} = \frac{hit}{total} = 0.9$ and 0.95 for the training phase and the adaptation phase respectively, where $total$ represents the number of times the agent completes the LTL task and hit indicates the number of times the current task is completed within one hundred steps. Fig. 3 shows the average normalized discounted reward across 10 unseen tasks with 5 random seeds in the adaptation phase. Clearly, MQMT outperforms these leading methods for sequential tasks.

2) Case 2. Interleaving Tasks: To test the flexibility of performing tasks, partial tasks are allowed to be completed in any order in this case. Taking φ_{apple} in Case 1 as an example, Dinning_Room and Kitchen can be visited in any order before going to Living_Room. Such a task can be rewritten as $\varphi_{\text{apple}} = \Diamond(\text{Dinning\_Room} \wedge \Diamond\text{Living\_Room}) \wedge \Diamond(\text{Kitchen} \wedge \Diamond \text{ Living\_Room})$. Fig. 4 illustrates the results after removing unnecessary orders from the sequential tasks in Case 1, which shows MQMT outperforms the leading methods in terms of convergence speed and rewards collection when encountering new interleaving tasks in the adaptation phase.

3) Completed Tasks: Fig. 5 summarizes the maximum completed tasks in the adaptation phase using different algorithms for all tasks in Case 1-2. Different color bars represent different LTL tasks in each case and its length represents the steps used to complete the corresponding task. Fig. 5 shows MQMT has converged for all tasks at around 20,000 steps. After that, MQMT requires no more than 1500 steps for task completion, even for challenging tasks. In contrast, other algorithms complete less than 10 tasks after 50,000 steps.

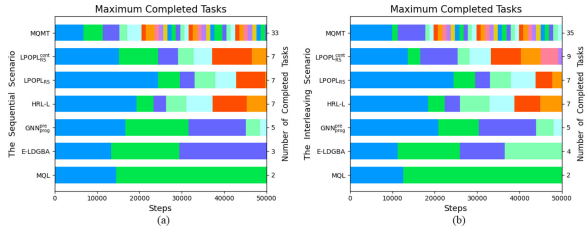

Fig. 5. Maximum completed tasks using different algorithms for sequential tasks in (a) and interleaving tasks in (b). Different tasks are represented by different colors and the length of each bar indicates the required steps to finish the corresponding task.

4) Discussions: Fig. 3(a) and Fig. 4(a) show that the context variable does help LPOPL_{RS} improve the performance. However, the performance of $\text{LPOPL}_{\text{RS}}^{\text{cont}}$ is still not comparable to that of MQMT. Although E-LDGBA has good performance for single LTL tasks, it learns the sub-tasks of multiple tasks individually and cannot utilize the reward to simultaneously learn other tasks. Thus, the performance of E-LDGBA degrades for multiple tasks. As to $\text{GNN}_{\text{prog}}^{\text{pre}}$, the GNN module greatly improves the representation of the LTL task, leading to improved performance of RL. However, if the reward is sparse in the environment, such a method lacks reward shaping to facilitate the learning of multiple tasks. In HRL-L, the option for the next goal is often executed after a sub-goal with some option is achieved. Although there might exist many goals with the same property, HRL-L only focuses on an optimal option for the current state, and thus result in local optimal policies. As a meta-RL algorithm, MQL outperforms classical RL in many problems. However, MQL still mainly focuses on single-objective problems and shows limited capability for complex multi-task problems. Fig. 3(b) and Fig. 4(b) show how the performance varies with different training steps. Apparently, 75,000 training steps show the best performance. The 10,000, 25,000 and 150,000 training steps show degraded performance mainly due to possible under-fitting or over-fitting issues.

B. Experiment Results

Experiments are also performed on a mobile robot, Turtlebot3 burger, for the sequential and interleaving tasks considered in Section V-A to verify the effectiveness of the developed MQMT. The experiment video and more experiment details are provided.[1]

[1][Online]. Available: https://youtu.be/QcDtNGs1V7I

VI. Conclusion

In this letter, we present a MQMT framework which incorporates meta-RL to facilitate the learning of multiple tasks and the generalization to new tasks. Additional work will consider extension to continuous workspace and investigate the issues of catastrophic forgetting, task blurring, and possible solution instability on previously learned tasks in meta-RL.

References

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

[2] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast reinforcement learning via slow reinforcement learning," 2016, *arXiv:1611.02779*.

[3] J. X. Wang *et al.*, "Learning to reinforcement learn," 2016, *arXiv:1611.05763*.

[4] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[5] Z. Xu, H. van Hasselt, and D. Silver, "Meta-gradient reinforcement learning," 2018, *arXiv:1805.09801*.

[6] R. Houthooft *et al.*, "Evolved policy gradients," 2018, *arXiv:1802.04821*.

[7] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5331–5340.

[8] R. Fakoor, P. Chaudhari, S. Soatto, and A. J. Smola, "Meta-q-learning," 2019, *arXiv:1910.00125*.

[9] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion," *IEEE Robot. Autom. Mag.*, vol. 14, no. 1, pp. 61–70, Mar. 2007.

[10] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT press, 2008.

[11] M. Cai, S. Xiao, Z. Li, and Z. Kan, "Optimal probabilistic motion planning with potential infeasible LTL constraints," *IEEE Trans. Autom. Control*, to be published, doi: 10.1109/TAC.2021.3138704.

[12] M. Cai, M. Hasanbeig, S. Xiao, A. Abate, and Z. Kan, "Modular deep reinforcement learning for continuous motion planning with temporal logic," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7973–7980, Oct. 2021.

[13] M. Cai, H. Peng, Z. Li, and Z. Kan, "Learning-based probabilistic LTL motion planning with environment and motion uncertainties," *IEEE Trans. Autom. Control*, vol. 66, no. 5, pp. 2386–2392, May 2021.

[14] X. Li, Z. Serlin, G. Yang, and C. Belta, "A formal methods approach to interpretable reinforcement learning for robotic planning," *Sci. Robot.*, vol. 4, no. 37, 2019, Art. no. eaay6276.

[15] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Teaching multiple tasks to an RL agent using LTL," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2018, pp. 452–461.

[16] Y.-L. Kuo, B. Katz, and A. Barbu, "Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2020, pp. 5604–5610.

[17] B. G. Leon, M. Shanahan, and F. Belardinelli, "Systematic generalisation through task temporal logic and deep reinforcement learning," 2020, *arXiv:2006.08767*.

[18] P. Vaezipoor, A. Li, R. T. Icarte, and S. McIlraith, "LTL2action: Generalizing LTL instructions for multi-task RL," 2021, *arXiv:2102.06858*.

[19] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith, "LTL and beyond: Formal languages for reward function specification in reinforcement learning," in*Proc. Int. Joint Conf. Artif. Intell.*, 2019, vol. 19, pp. 6065–6073.

[20] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *J. Artif. Intell. Res.*, vol. 73, pp. 173–208, 2022.

[21] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods Syst. Des.*, vol. 19, no. 3, pp. 291–314, 2001.

[22] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[23] F. Bacchus and F. Kabanza, "Using temporal logics to express search control knowledge for planning," *Artif. Intell.*, vol. 116, no. 1/2, pp. 123–191, 2000.

[24] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc. Assoc. Adv. Artif. Intell. Fall Symp. Ser.*, pp. 29–37, 2015.

[25] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*.

[26] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proc. Int. Conf. Mach. Learn.*, 1999, vol. 99, pp. 278–287.

[27] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, vol. 30, no. 1, pp. 2094–2100.

[28] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2052–2062.

[29] D. Agarwal, L. Li, and A. Smola, "Linear-time estimators for propensity scores," in *Proc. Int. Conf. Artif. Intell. Stat.*, 2011, pp. 93–100.

[30] V. Elvira, L. Martino, and C. P. Robert, "Rethinking the effective sample size," 2018, *arXiv:1809.04129*.

[31] R. Fakoor, P. Chaudhari, and A. J. Smola, "P3O: Policy-on policy-off policy optimization," in *Proc. Uncertainty Artif. Intell.*, 2020, pp. 1017–1027.

[32] M. Cai, S. Xiao, B. Li, Z. Li, and Z. Kan, "Reinforcement learning based temporal logic control with maximum probabilistic satisfaction," in *Proc. Int. Conf. Robot. Autom.*, 2021, pp. 806–812.

[33] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. 29th Conf. Neural Inf. Process. Syst.*, 2016, vol. 29.